A Real-time Implementation of Rendering Light Field Imagery for Generating Point Clouds in Vision Navigation

by

Christopher Rose

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 16, 2017

Keywords: plenoptic cameras, point clouds, CUDA, OpenMP

Approved by

Abstract

This dissertation develops a real-time implementation of rendering perspective and refocused imagery from a light field camera for the generation of sparse point clouds using a traditional stereo camera approach, a multiperspective approach, and two refocused ranging approaches. Unlike monocular cameras that have difficulty in extracting depth information without prior knowledge of a scene either in the form of previous images or object recognition, light field, or plenoptic, cameras can estimate depth with the introduction of additional hardware in the form of a microlens array to capture the 4-D radiance. The centers of these microlens are extracted with a calibration procedure and are used to build the radiance that contains the additional angular information that gives the plenoptic camera its ability to render multiple perspectives with a single main lens, thus improving upon the stereo or multiple camera setups that require more space and power as well as additional hardware for triggering synchronous images. This additional information for rendering multiple perspectives and refocused images greatly increases the plenoptic image size and thus the image processing time. With the end of dramatic growth in single processor performance, parallel processing has become the dominant method to decrease computing times. In response, this dissertation develops efficient methods for plenoptic image rendering algorithms for single core processor, multiple core processor, and graphics processing unit (GPU) architectures.

The multiple core processor architecture is realized through OpenMP with an Intel Core i7-4720HQ processor, and the GPU implementation uses Nvidia's CUDA API with a GeForce GTX 980M. For the perspective image rendering from the plenoptic case, OpenMP gives a 3.9x speedup from 21.8 ms with a single core to 5.6 ms on eight cores. The integral refocused image case improves its computation time with OpenMP from 699 ms for the serial architecture to 242.1 ms with eight cores for a 2.9x speedup. A 3.3x speedup is achieved with OpenMP for the FFT refocused imaging case from 717.3 ms for one core to 217.4 ms using eight cores. The perspective image rendering from the plenoptic image case has a GPU computation time of 5.6

ms and a speedup of 3.9x over the single core processing time. Computation with CUDA takes 17.3 ms to render an integral refocused image which amounts to a 40x speedup over the single core processing time. Finally, the FFT refocused imaging case results in a 14.6x speedup over the serial time with a total computation time of 49 ms.

Four methods are discussed in this dissertation for determining range using these efficiently rendered multiple perspective and refocused images: stereo ranging, multiperspective ranging, integral refocused ranging, and FFT refocused ranging. The stereo ranging approach follows similar techniques as traditional stereo algorithms through the ability of the plenoptic camera to render two virtual cameras. A multiperspective ranging approach uses an extrinsic property calibration model to rectify multiple virtual stereo pairs for a robust multiple stereo-pair ranging approach. Two depth-from-focus approaches use correlation to measure the depth through either an integral-based refocusing method or an FFT-based refocusing method to compare the α-refocused image with the in-focus perspective image. A ranging model then relates the α at the maximum correlation with the distance. During experimental testing, each of the four ranging algorithms estimates ranges to tracked features on a target using a 55 mm lens and a 135 mm lens. The 55 mm lens renders images with a field of view of 40° at the cost of a maximum range of about 2 meters, while the 135 mm lens estimates improved ranges out to 6.5 meters at the cost of a field of view of only 16°. Finally, this dissertation demonstrates plenoptic odometry with target tracking using each of the four ranging methods.

Acknowledgments

I would first like to thank all of the graduate students past and present in the GPS and Vehicle Dynamics Laboratory for their input and time over the years. They made graduate school that much more enjoyable. I would also like to thank my adviser Dr. David Bevly for his guidance throughout my time in graduate school. I also want to thank the Advanced Flow Diagnostics Laboratory and especially Tim Fahringer for allowing me to borrow their plenoptic camera. I would also like to express my appreciation to my family for their enthusiasm to see me graduate. Also, I would like to thank the folks at Leidos for their thoughtful input and allowing me the use of a navigation dataset.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Visual odometry is a well researched area with applications in robotics, navigation, controls, autonomy, augmented reality, and many other areas of study. One of the most difficult aspects of visual odometry is depth discrimination to determine the distance to an object. Traditional visual odometry algorithms use stereo cameras using triangulation or monocular camera algorithms using parallax over successive images to determine depth to features. This dissertation describes image feature ranging using a relatively new type of camera - the plenoptic camera. Plenoptic cameras improve upon standard cameras by capturing the light field entering the aperture through the use of a microlens array that is mounted directly in front of the sensor array. The light field includes angular information on the light striking the sensor array, and this angular information gives the ability to render multiperspective images and post-processed refocused images from a single photograph [1]. By combining the multiperspective images with traditional stereo and multi-camera algorithms, the depth to image features can be found for point cloud computation. Additionally, the refocused images provide depth as well by rendering images based on a simulated main lens at a scaled, α, distance from the focal length, which allows an entire focal stack of images to be created. Through knowledge of the relationship between the α variable and depth, the distance to a particular focused feature can be found, and a point cloud can be computed.

In order to capture the additional angular information, the sensor array must be large. The rendered images are a fraction of the size of the original plenoptic image from the camera, and processing this data can be a problem for real-time robotics. Due to the end of significant

Figure 1.1: A lensless plenoptic camera built through Imperx

growth in single processor performance [2], chip manufacturers have relied on parallel processing to continue increasing computational capability [3]. This dissertation explores three architectures for computing rendered images in real time: a serial case and two parallel architectures known as OpenMP and CUDA. OpenMP provides an API for implementing multithreaded applications, and CUDA is a framework for computing massively parallel tasks on an Nvidia Graphical Processing Unit. Execution times are presented that show the benefits and drawbacks of each of the architectures and algorithms. Results show a significant speedup over the serial case for image refocusing using parallel architectures, thus enabling real-time capability for refocused ranging with a plenoptic camera.

A traditional stereo odometry algorithm with plenoptic imaging uses two rendered plenoptic perspective images to simulate a stereo camera pair for determining range through triangulation. After stereo rectification, a block matching algorithm calculates the disparity between the stereo imagery for feature rich areas of the image. These features can be tracked from frame to frame to estimate motion from one timestamp to the next while providing stereo correspondence for determining depth.

Extending the perspective imagery to more than two images leads to multiperspective imagery for determining disparity at feature points. While geometric representations of three, four, and additional camera views exists in literature [4], the use of additional perspectives to determine disparity can increase robustness over the stereo method. With this multiperspective plenoptic method, multiple perspective images can be used to find their corresponding feature points through an automatic rectification model using the assumption of relative static virtual cameras generated from the (u, v) image plane. In this dissertation, the additional camera perspectives are rendered in a circular arrangement, which results in corresponding features that create a circular pattern as well. Using the mode of these corresponded points from a center image, the disparity can be found.

While the perspective images provide a means of providing stereo or multi-camera geometry, refocused plenoptic imagery gives another means of establishing depth from a single camera through depth-from-focus methods. A depth discrimination algorithm is presented that uses refocused imaging and prior knowledge of the relationship between the depth and the α term from camera geometry to generate a point cloud of the image features. Successive point clouds from the refocused image stack over time can then be used to calculate the rotation and translation in a standard visual odometry framework. Finally, an experimental demonstration of visual odometry from a plenoptic camera is shown, and the benefits and drawbacks to plenoptic visual odometry are described.

1.1 Contributions

Contributions of this dissertation broadly include efficient, real-time plenoptic image processing as well as the use of those image rendering techniques to generate point clouds for visual odometry. Specifically these contributions include:

- Computationally efficient plenoptic image rendering
  - Multi-core implementation using OpenMP for the three rendered image types
  - GPU implementation using CUDA for the three rendered image types
- Plenoptic ranging methods using rendered images
  - Development of a traditional stereo ranging algorithm using plenoptic imagery

  - Development of a multiperspective ranging algorithm using plenoptic imagery
  - Depth-from-focus algorithms using refocused plenoptic imagery

- A model for calculating the extrinsic properties for the full 4-D light field using only a subset of perspective images
- Ranging models for the integral and FFT refocused imagery
- Discussion of design and considerations for a plenoptic camera developed for visual odometry

1.2 Dissertation Outline

This dissertation begins with background on camera geometry, stereo camera geometry, and plenoptic camera geometry in Chapter 2. Chapter 3 gives the theoretical description of the three image rendering algorithms. Chapter 4 describes the implementation of the perspective and refocused imaging algorithms in the serial, multithreaded, and GPU architectures and provides the execution times for each method. Chapter 5 describes the stereo and plenoptic visual odometry methods that use the perspective and refocused rendered images, and Chapter 6 provides an experimental setup and results for these algorithms. Finally, Chapter 7 presents an overview of the results of the dissertation and gives ideas for relevant future work.

Chapter 2

Camera Models and Geometries

Any navigational algorithm requires understanding the relationship between different coordinate frames. For image processing, that relationship is defined through camera models that relate real world coordinates to image coordinates. The pinhole camera model relates the 3-D world coordinates to a single (monocular) camera, while epipolar geometry describes the relationship between two pinhole camera models (stereo) used for triangulation of world coordinate points to determine depth. The pinhole camera model relies on characteristics of cameras known as intrinsic properties, while epipolar geometry relies on transformation properties between the cameras known as extrinsic properties. Both the intrinsic and extrinsic properties are determined through preprocessing calibration procedures. The plenoptic camera, which is the principal focus of this dissertation, cannot be explicitly modeled as a pinhole camera, but virtual pinhole cameras can be created from the plenoptic imagery through image rendering techniques. This chapter describes the camera models used in this dissertation for relating real world coordinates to camera coordinates through the generation of images from multiple pinhole camera models through plenoptic imagery. The next section defines the pinhole camera model for monocular cameras.

2.1 Monocular Camera Geometry

Monocular cameras typically have a few common elements - the aperture, the main lens, and the sensor array. The aperture is the point at which light enters the camera. The main lens is typically made of glass and often has some way to move it along the optical axis for focusing. Finally, the sensor array is made of complementary metal-oxide semiconductor (CMOS) or charge-coupled device (CCD) arrays which can have millions of photosites arranged in a rectangular grid. Low-cost color filters are printed on the surface with one red, one blue, and two green filters. An image is then created from the light striking the sensor array [5]. This dissertation discusses the plenoptic camera that adds an additional microlens array directly in front of the sensor array. Images rendered from the plenoptic imagery result in virtual cameras that can be modeled as a pinhole camera. The next section describes the pinhole camera model.

2.1.1 The Pinhole Camera

One of the most common image processing models, shown in Figure 2.1, is the pinhole camera model where the center of projection is the point at which all rays from the world converge. While other models have been explored both in a monocular sense and a plenoptic sense [6], the pinhole model is the most common camera model used today. The pinhole camera is modeled as seen below in Equation (2.1) and Equation (2.2).

$$x = S_x \lambda \frac{X}{Z} + x_0 \tag{2.1}$$

$$y = S_y \lambda \frac{Y}{Z} + y_0 \tag{2.2}$$

where x and y are in the image coordinate system, x_0 and y_0 are the coordinates of the image center, λ is the camera focal distance, S_x and S_y are scale factor associated with the physical pixel dimensions, and $[X, Y, Z]$ are world coordinates given in the camera frame. $S_x\lambda$ and $S_y\lambda$ are more commonly represented as f_x and f_y, since the scale factors S_x and S_y are more difficult to determine than $S_x\lambda$ and $S_y\lambda$, which can be found through common calibration techniques. Replacing $S_x\lambda$ and $S_y\lambda$ results in Equations (2.1) and (2.2), which leads to the following perspective projection equations:

$$u = f_x \frac{X}{Z} + u_0 \tag{2.3}$$

$$v = f_y \frac{Y}{Z} + v_0 \tag{2.4}$$

where f_x and f_y are the effective focal distances for the x and y coordinates, respectively.

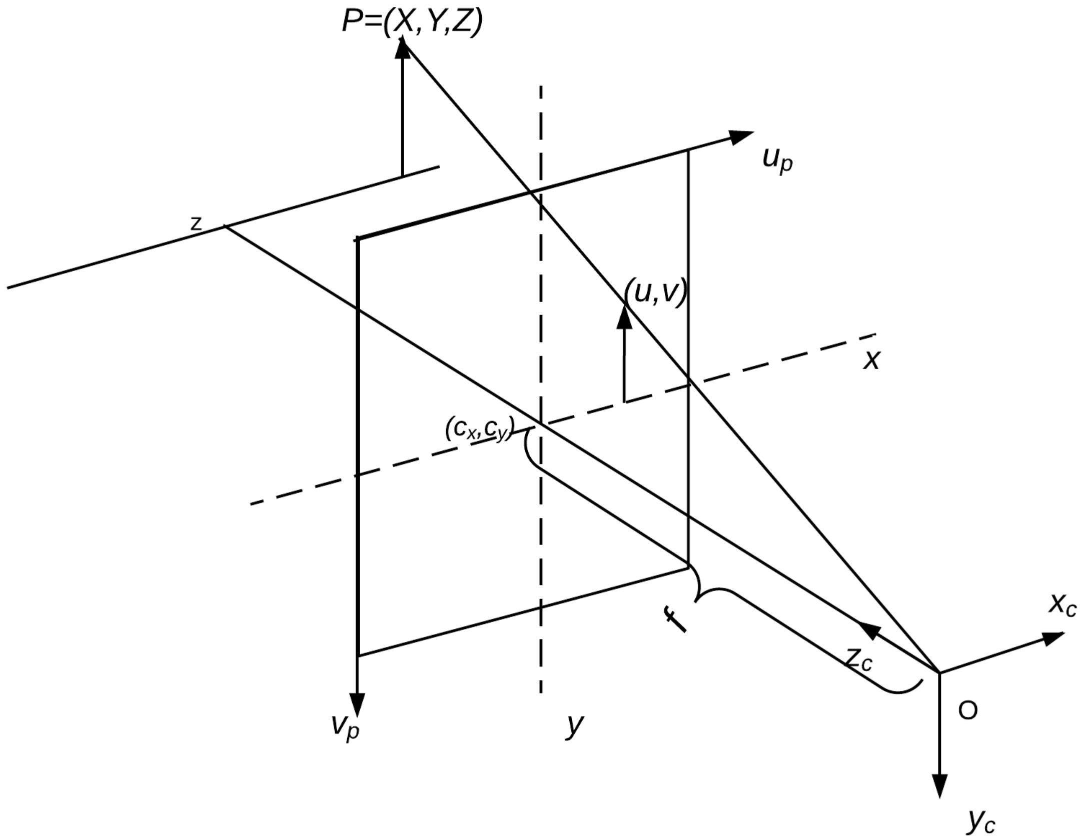


Figure 2.1: The pinhole camera model

The relation between a point, P, moving in the world and a point moving in the camera frame is the basis of understanding the motion of a camera from frame to frame. The motion of a camera in the world is described by Equation (2.5).

$$\dot{P} = \omega \times P + V \tag{2.5}$$

where ω is the angular velocity and V is the translational velocity of a camera. The relationship of the motion of a world point, $P = (X, Y, Z)$, and a point in a camera frame, (u, v), is shown in Equations (2.6), (2.7), and (2.8).

$$\dot{X} = z\omega_y - \frac{vz}{\lambda}\omega_z + V_x \tag{2.6}$$

$$\dot{Y} = \frac{uz}{\lambda}\omega_z - z\omega_x + V_y \tag{2.7}$$

$$\dot{Z} = \frac{z}{\lambda}(v\omega_x - u\omega_y) + V_z \tag{2.8}$$

where $\dot{X}$, $\dot{Y}$, and $\dot{Z}$ are the derivatives of the coordinates of P [7]. Using the quotient rule,

$$\dot{u} = \frac{f}{Z}T_x - \frac{u}{Z}T_z - \frac{uv}{f}\omega_x + \frac{f^2 + u^2}{f}\omega_y - v\omega_z \tag{2.9}$$

$$\dot{v} = \frac{f}{Z}T_y - \frac{v}{Z}T_z + \frac{-f^2 - v^2}{f}\omega_x + \frac{uv}{f}\omega_y - u\omega_z \tag{2.10}$$

The image Jacobian of Equation (2.11), then, relates the image space velocity of a point to the relative velocity of a world coordinate point.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{z} & 0 & \frac{-u}{z} & \frac{-uv}{f_x} & \frac{-(f_x^2+u^2)}{f_x} & -v \\ 0 & \frac{f_y}{z} & \frac{-v}{z} & \frac{f_y^2+u^2}{f_y} & \frac{uv}{f_y} & u \end{bmatrix} \times \begin{bmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2.11}$$

where u and v are the coordinate system for the aforementioned virtual image plane, f_x and f_y are the camera focal lengths, x-y-z are the aforementioned world coordinates, V_x-V_y-V_z are the translation motion in the x-y-z axis, and ω_x-ω_y-ω_z are the angular velocities. Notice that the range, z, is present in Equation (2.11), which represents the effect of range on the velocity of a point in the image. Since range is not directly determined from an image, slow near objects will seem to move at the same speed as distant fast objects. In addition, the image plane velocity is dependent on six motion components, which makes it impossible to distinguish rotational from translational motion from a single point. Fortunately, two (or stereo) cameras can be used to calculate range, which is similar to a human's eyes. However, humans use more than just stereo triangulation to perceive range. Humans use nine visual cues to perceive distance, such as determining which objects are closer and prior knowledge of the sizes of objects to perceive their distance [8].

Visual navigation systems typically use either one (monocular) or two (stereo) cameras. Although camera systems with more than two systems have been used [9], they are not often employed in consumer applications due to the additional cost and processing requirements in multi-camera systems. The benefit of two cameras over a single camera is typically for estimating distances in order to discern the difference between fast moving far landmarks and slow moving near landmarks.

2.1.2 Homogeneous Coordinates

The concept of points or lines at infinity is commonly used in image processing to describe the vanishing distance or parallel lines in an image. While Euclidean coordinates are often used to represent 3-D and 2-D coordinates, they have no means to represent a point at infinity such as the intersection of parallel lines. Homogeneous coordinates, however, can represent infinity through the addition of an additional coordinate W. With the addition of a new coordinate, a 3-D point in Euclidean point is represented as a 3-D vector in homogeneous coordinates. Similarly, homogeneous coordinates in 2-D are represented by the homogeneous vector tuple (X, Y, W). Each homogeneous coordinate of a point are related by the equivalence class, or when vectors are equivalent when they differ by a multiple. The fourth coordinate W represents this multiple, since the ray (kX, kY, kZ, k) maps to the same point as $(X, Y, Z, 1)$. Standard representation of the homogeneous coordinate typically divides each coordinate by the fourth coordinate as shown in Equation (2.12).

$$X = (\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}, 1) \tag{2.12}$$

In the special case where $W = 0$, each coordinate in point X goes to infinity. Point X, then lies at infinity. In 2-D coordinates, this point lies on the line at infinity. In 3-D coordinates, the point lies on the plane at infinity. Thus, any point, $(X, Y, Z, 0)$ or $(X, Y, 0)$ whose last homogeneous coordinate is 0, lies at infinity [4].

Homogeneous coordinates give rise to projective geometry, wherein points at infinity are no different from other points. As such, any point at infinity does not necessarily stay at infinity in a linear transformation. Therefore, the concept of parallel lines is not relevant in projective geometry - parallel lines merely intersect at a $(X, Y, Z, 0)$ point rather than a (X, Y, Z, W) point where $W \neq 0$. A projective transform, then, can be represented by a linear transform as seen in Equation (2.13).

$$X' = H_{4x4}X \tag{2.13}$$

In imagery, horizontal parallel lines meet at the vanishing point on the horizon line. By extending homogeneous coordinates to treat parallel lines as any other lines, a relationship exists between points in an image and points on the world plane, regardless of whether the points lie at infinity or not [4].

2.1.3 Representation of an Image Point as a Line of 3-D Points

A point in an image represents the line from the camera center to an object at P. The camera center can be represented as the null vector of the ray OP,

$$P_{CM}C = 0 \tag{2.14}$$

where P_{CM} is the camera matrix. The 3-D point, P, can be represented as

$$P = P_{CM}^{+}x \tag{2.15}$$

where P^+ is the pseudoinverse of P as given in the following equation.

$$P^+_{CM} = P^{\intercal}_{CM}(P_{CM}P^{\intercal}_{CM})^{-1} \tag{2.16}$$

The ray is the line formed by the mathematical join of the point P and the camera center as given in Equation (2.17).

$$P(\lambda) = P^+_{CM}x + \lambda C \tag{2.17}$$

2.1.4 The Camera Matrix

Projective geometry models imagery as a 2-D projective plane ((X, Y, W) in homogeneous coordinates), while the real world coordinates are modeled as the 3-D projective plane ((X, Y, Z, W) in homogeneous coordinates). Any point $(X, Y, Z, 1)$ is equivalent to the point (X, Y, Z) in Euclidean coordinates. The transform from 3-D homogeneous coordinates to 2-D homogeneous coordinates consists of a 3×4 matrix K_{CM}, called the camera matrix, as shown in Equation (2.18).

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = K_{CM} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{2.18}$$

Inherent ambiguities in the camera coordinate frame, the camera properties itself, and the direction the camera currently points must also be taken into consideration. The camera properties are described by the camera intrinsic matrix, K_{int}, which contain the focal lengths, f_x and f_y, the coordinates of the image centers, c_x and c_y, and a skew parameter γ, which is hereafter considered 0.

$$K_{int} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

The direction of the camera and the inherent ambiguities in the camera frame are presented by the extrinsic matrix of the camera, K_{ext}. The rotation parameters, r, and translation parameters, t, define the location and direction of the camera in the world.

$$K_{ext} = [R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.20}$$

where the rotation matrix, R, and the translation vector, t, define the transform from the world coordinates, (X, Y, Z), to a coordinate system, (x, y, z) fixed to the camera.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{2.21}$$

The camera matrix K_{CM} describes the transform between image coordinates (u, v) and world coordinates (X, Y, Z) using the internal parameters (f_x, f_y, c_x, c_y) and the external parameters (r, t).

$$K_{CM} = K_{int}K_{ext} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.22}$$

Note that the intrinsic camera matrix, K_{int}, is sometimes simply referred to as the camera matrix.

2.2 Epipolar Geometry

Epipolar geometry describes the geometric relationship between two cameras. With known epipolar geometry, the location of features in one camera can be found along a line in the image of another camera. This line is known as an epipolar line, which consists of the intersection of the epipolar plane with the image plane. The epipolar plane is any plane containing the baseline of the stereo pair; the epipolar planes for a particular baseline make up a pencil of epipolar planes. The epipole is the intersection of the baseline with the image plane. For cameras pointing at each other, the epipole is the point consisting of the center of the other camera. For this dissertation, the epipole will not be visible in any image since none of the virtual cameras are capable of pointing at each other.

Figure 2.2 shows two vectors, P_L and P_R, that represent the rays from L_0 to X and R_0 to world point X, respectively. The translation vector T represents the baseline the two cameras. The TP_LP_R plane is an epipolar plane that contains the epipolar line e_l' in the right image and passes through points e and x in the left image. The points e and e' are epipoles. The epipolar geometry between two cameras and the underlying epipolar model uses two important matrices - the essential matrix and the fundamental matrix.

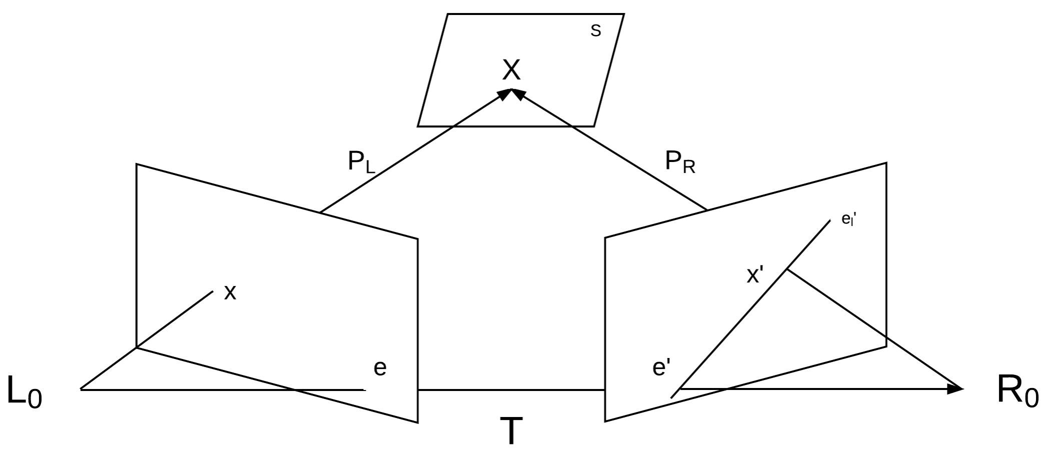


Figure 2.2: Epipolar geometry

2.2.1 The Fundamental Matrix

The fundamental matrix describes the epipolar geometry of two views using image points to describe the stereo geometry.

2.2.1.1 Geometric Derivation

Corresponding points in each image are related through a homography, H_s, that consists of any plane s that holds the 3-D point X in Figure 2.2. This homography allows for the transformation of x points to x' points.

$$x' = H_s x \tag{2.23}$$

The epipolar line e'_l consists of the cross product of the point e' from the baseline and the point x' from the projection line P_R.

$$e'_l = e' \times x' = [e']_\times x' \tag{2.24}$$

where the notation $x_\times$ acts as a skew-symmetric matrix on a vector $x = [x_1, x_2, x_3]^\intercal$ as described in Equation (2.25).

$$x_\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{2.25}$$

Using Equation (2.23), Equation (2.24) becomes

$$e'_l = e' \times H_s x = [e']_\times H_s x \tag{2.26}$$

Equation (2.26) defines the transform from a point x to the epipolar line e'_l. The fundamental matrix is defined as this transform [4].

$$F = e' \times H_s = [e']_\times H_s \tag{2.27}$$

2.2.1.2 Algebraic Derivation

Since image points are used as part of the fundamental matrix, the intrinsic properties of the cameras can be incorporated into the derivation of the fundamental matrix. The algebraic derivation begins with the projection of the point X in Figure 2.2 onto the left camera.

$$x = P_L X \tag{2.28}$$

The equation of the ray back-projected from x by P_L is given in Equation (2.17). At $\lambda = 0$, Equation (2.17) defines the point $P_L^+ x$, while $\lambda = \infty$ describes the left camera center. In the right camera, the left camera center is described as

$$L_0 = X(0) = P_R P_L^+ x \tag{2.29}$$

while the point x is

$$X(\infty) = P_R L_0 \tag{2.30}$$

The epipolar line for these two points is

$$e_l' = (P_R L_0) \times (P_R P_L^+ x) \tag{2.31}$$

Note that the point $P_R L_0$ is merely the epipole in the right camera image, e'. Therefore,

$$e_l' = e' \times (P_R P_L^+ x) \tag{2.32}$$

Following the nomenclature of Equation (2.25), Equation (2.24) can be written as

$$e_l' = [e']_\times P_R P_L^+ x = Fx \tag{2.33}$$

where the fundamental matrix is defined as

$$F = [e']_\times P_R P_L^+ \tag{2.34}$$

which is identical to Equation (2.27) for $H = P_R P_L^+$, where H is defined in terms of the camera matrices. In this derivation, if $R_0 = L_0$, $F = \boldsymbol{O}$. Incorporating the rotation and translation matrices into Equation (2.34) gives the following equations for the fundamental matrix.

$$F = K'^{-\intercal}[T]_\times R K^{-1} = K'^{-\intercal} R [R^\intercal T]_\times K^{-1} \tag{2.35}$$

These equations describe the effect of the rotation and translation of the cameras on the fundamental matrix [10, 4].

2.2.2 The Essential Matrix

The essential matrix describes the epipolar geometry of two cameras through the extrinsic parameters, R and T, of the two cameras. Following the nomenclature of Figure 2.2, the vector P_R can be defined as given in Equation (2.36).

$$P_R = R(P_L - T) \tag{2.36}$$

where

$$P_L = K_{int}[R|T] \tag{2.37}$$

where $x = P_L X$. Since the intrinsic parameters are not needed for the essential matrix, the normalized coordinate $\hat{x}$ can be used to remove the intrinsic parameter from Equation (2.37).

$$\hat{x} = [R|T]X \tag{2.38}$$

where $[R|T]$ is the normalized camera matrix and $K_{int} = \boldsymbol{I}$. Assuming two normalized camera matrices, Equation (2.35) becomes

$$F_{K=\boldsymbol{I}} = E = [T]_\times R = R[R^\intercal T]_\times \tag{2.39}$$

Thus, the essential matrix, E, can be considered a form of the fundamental matrix, where the intrinsic camera matrix, K, is an identity matrix and the homogeneous coordinates are normalized. Calculating the essential matrix from the fundamental is straightforward by including the intrinsic matrices into Equation (2.35) as follows:

$$E = K'^\intercal F K \tag{2.40}$$

2.3 Camera Calibration

One of the basic requirements for calibrating any image navigation algorithm is the calibration of the intrinsic and extrinsic parameters of the camera. This section discusses intrinsic calibration of a pinhole model camera as well as extrinsic calibration of a stereo camera system.

2.3.1 Intrinsic Calibration

Intrinsic calibration is an integral part of using any kind of camera for navigation algorithms. Intrinsic calibration is used to estimate the camera matrix described in Section 2.1.4 as well as the lens distortion parameters. Intrinsic calibration is based on estimating the homography between a calibration model and its image. The homography consists of the transform required to fit the camera intrinsic parameters to the image and model and the distortion parameters required to warp the image to ensure parallel lines in the model are parallel in the image. The intrinsic calibration procedure minimizes the total reprojection error for all of the points in all of the given views from a camera. The most commonly used intrinsic calibration approach uses a checkerboard pattern as the known model, as described in [11] and [12]. AprilCal uses unique image fiducials for more robust detection [13]. For this work, an asymmetric circle pattern was used to extract circle centers rather than the chessboard square edges, since the number of required poses is less for an asymmetric pattern than a chessboard pattern for building a well-posed system of equations for calibration. The intrinsic parameters of the camera were previously described in Equation (2.19). The next section describes the distortion parameters used in intrinsic calibration.

2.3.1.1 Distortion Parameters

A relation between a point in the world and that point in the image is important. However, some cameras use imperfect lenses to reduce costs. These lenses contain distortion that alters the expected pixel location of a 3-D point through the pinhole camera model. For this reason, distortion in an image can be a significant source of error in a vision navigation system. Two types of distortions are common in images - radial and tangential. Radial distortion is the primary source of distortion in a cheap camera, while the tangential distortion is due to misalignments between the internal parts of the camera such as the lenses and sensor array. Intrinsic calibration, then, is designed to reduce this distortion. Most calibration algorithms warp the image with another distortion which negates the effect of the original distortion, creating a nondistorted image. One often-used example of this is the plumb bob model by Brown [14], which corrects distortion with the radial (k_1 , k_2 , k_3) and tangential (p_1 , p_2) components determined from a calibration procedure based on planar objects [15].

$$x_u = (x_d - x_c)(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2(x_d - x_c)^2 + 2p_2(x_d - x_c)(y_d - y_c)) \quad (2.41)$$

$$y_u = (y_d - y_c)(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_2(r^2 + 2(y_d - y_c)^2 + 2p_1(x_d - x_c)(y_d - y_c)) \quad (2.42)$$

Other distortion parameters include the small prism model and angular distortion due to the small rotation of the image sensor used to focus on an oblique plane in front of the camera, which is called the Scheimpfug condition [16]. Neither of these distortion parameters are used in this dissertation.

2.3.2 Extrinsic Calibration

In a stereo camera system, the location of an object in one camera is compared with the location of that same object in another camera. From this disparity, depth can be determined. However, any rotation or misalignment between the cameras can result in error in determining the corresponding locations between the images. Another calibration procedure, known as extrinsic calibration, is used to determine the rotation and translation between the two cameras by using corresponding points between two images. Like intrinsic calibration, it minimizes the total reprojection error for all of the points in all of the views from both cameras. One of the most popular calibration methods is Bouquet's method, unpublished but found in [12], which is based on previous work by Tsai [17] and Zhang [18, 11] in which previously intrinsic calibrations are used in the calibration procedure as the initial condition for the reprojection error minimization procedure. Hartley [19] also developed a method for calibration, but that method does not work well for images with high distortion [20].

2.4 The Plenoptic Camera

Researchers have been intrigued by the plenoptic camera for decades. In 1939, the first exploration into plenoptic cameras started with a description of the light field [21]. However, due to limitations in technology and processing capability, much of this early research was not practical for rendering images. In 1992, Adelson and Wang [22] used a lenticular array at the sensor location to calculate depth similarly to a stereo camera using a plenoptic camera. Gortler called the 4-D plenoptic function a lumigraph to render images of an object from new camera positions [23]. Due to the limitations of technology at the time, plenoptic technology stayed largely dormant until Ng's dissertation in 2006 [1].

Since then, plenoptic cameras have been used in many fields. Three dimensional particle image velocimetry is one such field in which the structure of turbulent flows is experimentally determined [24, 25]. In this field, high frame rate imagery can be used to determine the noise generated by high acoustic nozzles in military aircraft [26]. Some preliminary work regarding depth map estimation can be found in work by Carpenter [27]. In this work, Carpenter outlined several ranging algorithms including a stereo image method, a least squares gradient method, and common problems with obtaining ranges with a plenoptic camera. Other work involving stereo and multiperspective ranging with the plenoptic camera can be found in a paper by Roberts [28]. This work uses a correlation-based approach on the perspective imagery. This dissertation also uses a correlation-based approach except it uses the perspective image with the refocused image instead of another perspective image.

2.4.1 The Plenoptic Function

Plenoptic cameras are a relatively new technology which uses a lens array to capture the light field [1]. The purpose of any camera is to describe the world, and one method of doing so is with the so-called appearance description of the world, which consists of a dense array of light rays observed by a camera [29]. The plenoptic function, as defined by Adelson and Bergen [30], describes the world as a 7-D function consisting of the position of the camera X, Y, and Z, the azimuth ϕ and elevation θ of the light ray, any wavelength λ, and time t.

$$l^7 = (X, Y, Z, \phi, \theta, \lambda, t) \tag{2.43}$$

A 7-D function is infeasible for any practical application both due to memory requirements and for recording the data. In fact, no one has been able to sample the full plenoptic function in a single representation. Fortunately, by making some reasonable assumptions, the dimension of the plenoptic function can be reduced. The first assumption is to restrict the wavelength of the view to the visual range, which reduces the dimension of the plenoptic function by one. The second assumption states that the air is transparent and that the radiances along a light ray remains constant. With these two assumptions, the plenoptic function has been reduced to a 5-D function and is called light field video.

$$l^5 = (X, Y, Z, \phi, \theta) \tag{2.44}$$

In free space where the light rays do not change over time unless blocked, the 5-D plenoptic function is reduced to a 4-D function called the light field. The 5-D plenoptic function can be reparameterized to a 4-D function in free space and becomes the standard representation used for plenoptic cameras in this dissertation [29]. This reparameterization consists of defining each light ray as an intersection of two planes - one at infinity and one at the standard image plane. Setting the plane at infinity (and as such utilizing homogeneous coordinates as described in Section 2.1.2) allows two of the parameters to define direction (u, v), while the other two define a point (s, t) [31].

$$l^4 = (s, t, u, v) \tag{2.45}$$

Extending the plenoptic function to standard images requires the addition assumption that the viewer is in a fixed position, which reduces the dimension of the camera by three. With this assumption, the second assumption is irrelevant, which means a standard camera image is represented as follows in Equation (2.46) [29].

$$l^2 = (\phi, \theta) = (x, y) \tag{2.46}$$

As shown in Equation (2.46), standard cameras record a 2-D slice of the 4-D plenoptic function and are typically parameterized through a light rays intersection with the image plane at a pixel location (x, y).

2.4.1.1 Drawbacks

While the plenoptic camera allows the production of both perspective imagery and refocused imagery, those capabilities do not come without a cost. The rendered images are significantly smaller in resolution than other standard images, despite the sensor size of the camera. Processing the plenoptic camera requires a much significantly heavier burden for their resolution as well. Much of the raw plenoptic imagery is wasted in the portions of the images not seen by the microlens, which essentially have a pixel value of 0. The baseline of the virtual cameras is small for determining depth, and therefore many applications, such as vehicular navigation, are not feasible for current plenoptic cameras due to the lack of depth discrimination.

However, sensor array sizes are continually getting larger as camera technology improves and driven in the consumer desire for higher resolution images. This improvement will increase both the virtual baseline as well as the resolution of the rendered images.

2.4.1.2 The focused plenoptic camera

Another plenoptic camera was designed to improve upon some of these drawbacks of the plenoptic camera used in this dissertation. This new camera system is called plenoptic 2.0, or the focused plenoptic camera [32]. While the plenoptic camera used in this dissertation has a resolution based on the number of microlens, the focused plenoptic camera can have more than one pixel per microlens [33]. Computing correspondence is unnecessary since the camera parameters provide the necessary data [33]. Unfortunately, the focused plenoptic camera samples the lightfield with reduced angular density in exchange for increased spatial resolution

[32]. This decreased angular density will in turn reduce the virtual camera baseline even further. Note that the focused plenoptic camera is not used in this dissertation, which only uses a plenoptic 1.0 camera.

Chapter 3

Image Rendering

Rendered images from the plenoptic camera contain the 4-D lightfield with light not only from parallel light rays but also angular light rays due to the presence of the microlens array. Three image rendering methods are described in this section: two methods for rendering refocused images and one method for perspective imaging. The base data structure used for rendering these images is the radiance image L_F. First, however, the microlens centers of the raw plenoptic image, seen in Figure 3.1, must be found to properly construct the radiance image.

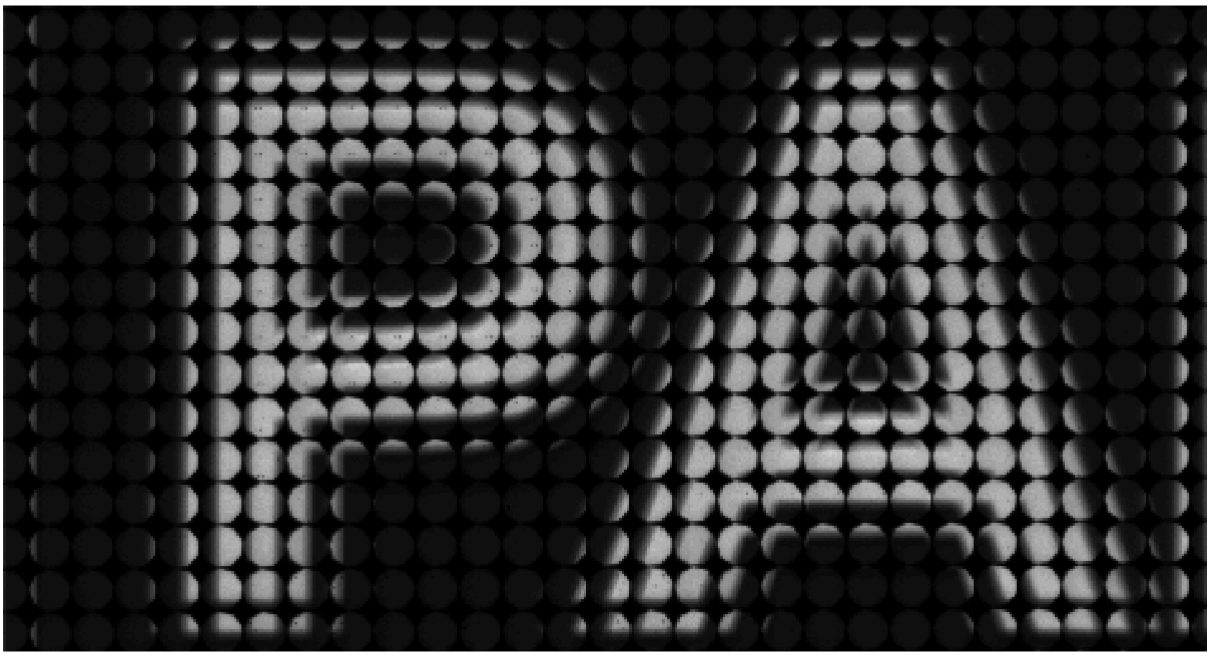


Figure 3.1: Microlens array in plenoptic image

3.1 Microlens Center Calibration

The raw plenoptic image for the plenoptic image used in this dissertation contains a rectangular array of circular microlens, which are the images of the lens entrance pupils onto the sensor. The 4-D lightfield is constructed from these circular microlens images, also known as

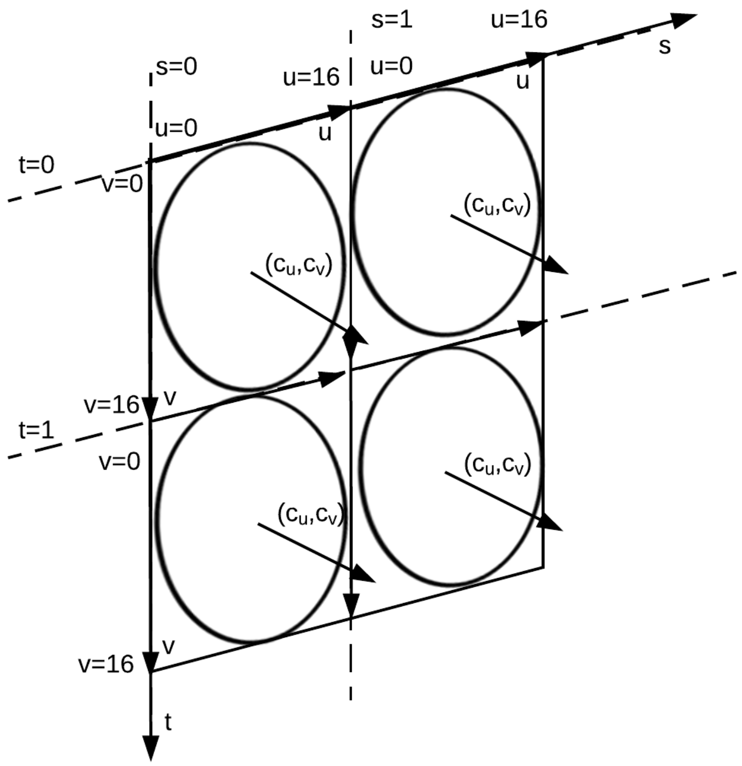


Figure 3.2: The 4-D coordinate system for the light field

elemental images, where the first two dimensions (s, t) consists of the index in the array while the last two indices (u, v) are the indices within the circular microlens, as shown in Figure 3.2. Due to errors in the mounting of the microlens array, the microlens are misaligned with respect to the image sensor array. The locations of the image centers must be found using a calibration image where each microlens center is illuminated from increasing the f-stop of the camera until the aperture is very small. Figure 3.3 shows small green dots at the locations of the microlens centers for the top left and the bottom left of the raw plenoptic image. The small rotation, θ, of the microlens array is clear. This rotation is also evident across the rows of the image and can cause the bottom row of the microlens array to drop off of the sensor plane. The resolution of the resulting plenoptic image is decreased due to these lost rows.

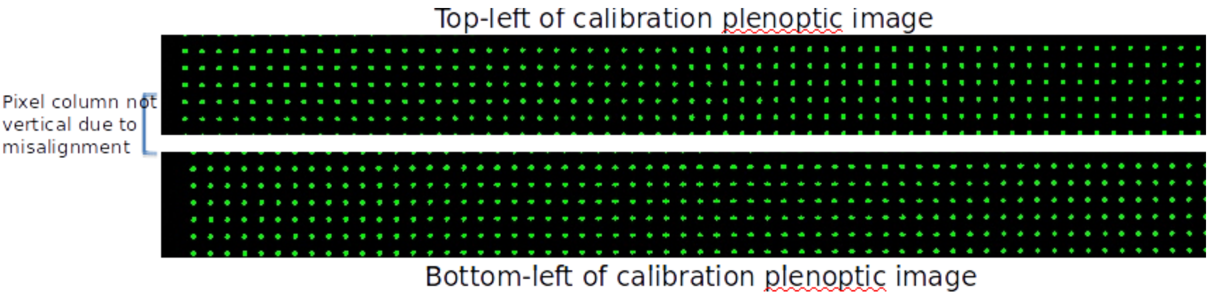


Figure 3.3: Microlens center misalignment

3.1.1 Marching Algorithm

A marching algorithm has been developed in this dissertation that calculates the misalignment θ of the microlens array from the image sensor. Following the detection of two image centers, the algorithm "marches" down while seeking additional microlens centers. After reaching the end of the calibration image, the algorithm searches down the next column of pixels in the calibration image.

3.1.1.1 Microlens Center Calibration- First Column

Determining the location of the first microlens center requires searching down each column for a pixel value greater than some heuristically chosen threshold to find the cluster of illuminated pixels that make up the microlens center. Once a pixel over the threshold is found, a small rectangular region of interest is used to calculate the subpixel microlens center (c_u, c_v).

3.1.1.1.1 Subpixel Microlens Center

Each image center consists of a small bead of light that strikes the sensor. The bead of light that is measured in the calibration image can be modeled as a Gaussian distribution of grayscale pixels, where the peak of the distribution lies at the center of the microlens array. Due to the discrete nature of images, however, the Gaussian distribution is sampled at each pixel, where the microlens center does not match up with a single pixel, as seen in Figure 3.4.

Thus, the coordinates of the microlens center is most likely at subpixel coordinates, where those subpixel coordinates are calculated from the pattern of grayscale images at the bead of light. Equations (3.1) and (3.2) show the calculation of the coordinates for the microlens center from the rectangular region of interest shown in Figure 3.4.

$$c_x = \frac{\sum_i^{rows} \sum_j^{cols} I(i,j)}{\sum_i^{rows} \sum_j^{cols} I(i,j) * i} \tag{3.1}$$

$$c_y = \frac{\sum_i^{rows} \sum_j^{cols} I(i,j)}{\sum_i^{rows} \sum_j^{cols} I(i,j) * j} \tag{3.2}$$

where i is the i^{th} row in the region of interest, j is the j^{th} column in the region of interest, and I is the image. Note that out of image pixels are ignored.

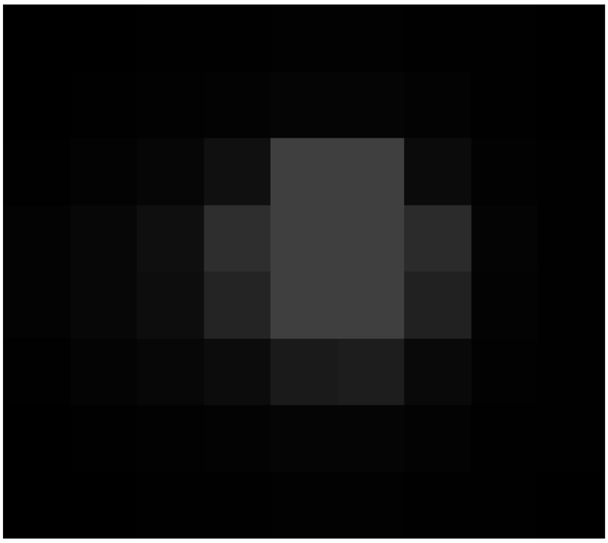


Figure 3.4: Microlens center in plenoptic image

3.1.1.1.2 Calculation of θ

Once the first microlens center is found, the search continues while ignoring the region outside of the neighborhood of the first microlens center in order to avoid detecting the same microlens center. Once the second microlens center is found, an initial estimate for the angle offset θ can be found using the equation below

$$\theta_0 = \arctan\left(\frac{C(0,0)_x - C(1,0)_x}{C(0,0)_y - C(1,0)_y}\right) \tag{3.3}$$

where $C(0,0)$ is the first microlens center point and $C(1,0)$ is the second microlens center point. This initial estimate speeds up the search for new microlens centers, since not only is the distance between microlens centers known, but the initial estimate for the angular offset θ is known as well. An estimate for the new location of the microlens can be found using Equations (3.4-3.5)

$$\hat{c}_x = C(0,0)_x + j * w_h * \sin(\theta) \tag{3.4}$$

$$\hat{c}_y = C(0,0)_y + j * w_h * \cos(\theta) \tag{3.5}$$

where $\hat{c}$ is the estimate of the location of the next microlens center, j is the index of the estimated microlens, and w_h is the height of the microlens. At this new location c_{xy}, a new microlens center is estimated within a small region of interest using the center of mass calculation. For each new estimated microlens center, the θ term is recalculated with the location of the new microlens center, as seen in Equation (3.6).

$$\theta = \arctan(\frac{C(0,0)_x - C(j-1,0)_x}{C(0,0)_y - C(j-1,0)_y}) \tag{3.6}$$

The algorithm proceeds down the first column until the last microlens is found. At this point, the θ term consists of the angle from the first microlens center at the top left of the image to the last microlens center in its column in the bottom left of the image.

3.1.1.2 Microlens Center Calibration - Columns

With the locations of the microlens centers of the first column as well as the angular offset θ known from the previous section, the microlens center calibration algorithm proceeds for the remaining columns. Each estimate for the microlens centers is repeated, but with the marching approach taken across the columns rather than the rows, as seen in Equations (3.7-3.8).

$$\hat{c}_x = C(i,j)_x + j * w_h * \cos(\theta) \tag{3.7}$$

$$\hat{c}_y = C(i,j)_y - j * w_h * \sin(\theta) \tag{3.8}$$

With the estimate of each microlens center, the center of mass calculation further refines the microlens center to subpixel accuracy, and all microlens centers are found.

The marching algorithm provides the locations of the microlens centers, but only needs to be run once, if the microlens centers' locations are not already known. Due to insufficient mounting (such as using glue) of the microlens array or mulitple lens, the marching algorithm may need to be run occasionally as the microlens array shifts with respect to the sensor array. Since the locations of the microlens' centers are assumed to be already known through the offline calibration procedure described in this section, this algorithm is ignored in the later discussions on computational efficiency.

3.2 Radiance Generation

The 4-D radiance image L_F contains all of the information from the raw image but is interpolated to compensate for the subpixel location of the microlens center. The radiance is generated with Equation (3.9)

$$\begin{aligned} L_{FR}(s,t,u,v) &= N_2(c_x(s,t) - r + u, c_y(s,t) - r + v, I) \\ L_{FI}(s,t,u,v) &= 0 \end{aligned} \tag{3.9}$$

where N_2 is bilinear interpolation over a plaid 2-D grid, I is the image, c_x and c_y are the coordinates of the microlens center, and s, t, u, and v are the subpixel coordinates of interest. In this equation, the coordinate pair (s,t) represents the microlens in view, while the coordinate pair (u,v) represents the pixel location on that microlens image. To facilitate operations in the Fourier domain, the data structure containing the radiance image can consist of a real part L_{FR} and an imaginary part L_{FI}. The next section describes the bilinear interpolation equation, N_2, in Equation (3.9).

3.2.1 Linear Interpolation

The fundamental linear interpolation algorithm for one dimension is shown in Equation (3.10)

$$N_1(x, I_1, I_2) = (1 - x) * I_1 + x * I_2 \tag{3.10}$$

where I_1 and I_2 are the grayscale values at known pixel locations.

The bilinear interpolation method of Equation (3.11) consists of two linear interpolations in one dimension, followed by the interpolation at the interpolated points in the second dimension. These equations assume a plaid, uniform grid over which the interpolation is calculated. Each point on the plaid grid has a corresponding value, and the distance to neighboring points is constant as shown below

$$N_2(x, y, I) = N_1(x - \lfloor x \rfloor, z_0(x, y, I), z_1(x, y, I)) \tag{3.11}$$

where the intermediate 1-D interpolation values z_0 and z_1 are shown in Equations (3.12) and (3.13).

$$z_0(x, y, I)) = N_1(y - \lfloor y \rfloor, I(\lfloor x \rfloor, \lfloor y \rfloor), I(\lceil x \rceil, \lfloor y \rfloor)) \tag{3.12}$$

$$z_1(x, y, I) = N_1(y - \lfloor y \rfloor, I(\lfloor x \rfloor, \lceil y \rceil), I(\lceil x \rceil, \lceil y \rceil)) \tag{3.13}$$

The `floor` and `ceiling` functions are designated here by $\lfloor\rfloor$ and $\lceil\rceil$, respectively.

The bilinear interpolation algorithm can be extended into four dimensions by further interpolating additional dimensions into fewer dimensions. Each dimension, then, is compressed into the next until the basic linear interpolation algorithm gives the final interpolated value. Quadrilinear interpolation, then, requires fifteen 1-D interpolation steps, which are represented in Equation (3.14).

$$\begin{aligned}
z_0(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lfloor t\rfloor, \lfloor u\rfloor, \lfloor v\rfloor), L_F(\lceil s\rceil, \lfloor t\rfloor, \lfloor u\rfloor, \lfloor v\rfloor)) \\
z_1(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lceil t\rceil, \lfloor u\rfloor, \lfloor v\rfloor), L_F(\lceil s\rceil, \lceil t\rceil, \lfloor u\rfloor, \lfloor v\rfloor)) \\
z_2(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lfloor t\rfloor, \lceil u\rceil, \lfloor v\rfloor), L_F(\lceil s\rceil, \lfloor t\rfloor, \lceil u\rceil, \lfloor v\rfloor)) \\
z_3(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lceil t\rceil, \lceil u\rceil, \lfloor v\rfloor), L_F(\lceil s\rceil, \lceil t\rceil, \lceil u\rceil, \lfloor v\rfloor)) \\
z_4(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lfloor t\rfloor, \lfloor u\rfloor, \lceil v\rceil), L_F(\lceil s\rceil, \lfloor t\rfloor, \lfloor u\rfloor, \lceil v\rceil)) \\
z_5(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lfloor t\rfloor, \lceil u\rceil, \lceil v\rceil), L_F(\lceil s\rceil, \lfloor t\rfloor, \lceil u\rceil, \lceil v\rceil)) \\
z_6(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lceil t\rceil, \lfloor u\rfloor, \lceil v\rceil), L_F(\lceil s\rceil, \lceil t\rceil, \lfloor u\rfloor, \lceil v\rceil)) \\
z_7(s,t,u,v,L_F)) &= N_1(s-\lfloor s\rfloor, L_F(\lfloor s\rfloor, \lceil t\rceil, \lceil u\rceil, \lceil v\rceil), L_F(\lceil s\rceil, \lceil t\rceil, \lceil u\rceil, \lceil v\rceil)) \qquad (3.14) \\
z_8(s,t,u,v,L_F)) &= N_1(t-\lfloor t\rfloor, z_0, z_1) \\
z_9(s,t,u,v,L_F)) &= N_1(t-\lfloor t\rfloor, z_2, z_3) \\
z_{10}(s,t,u,v,L_F)) &= N_1(t-\lfloor t\rfloor, z_4, z_5) \\
z_{11}(s,t,u,v,L_F)) &= N_1(t-\lfloor t\rfloor, z_6, z_7) \\
z_{12}(s,t,u,v,L_F)) &= N_1(u-\lfloor u\rfloor, z_8, z_9) \\
z_{13}(s,t,u,v,L_F)) &= N_1(u-\lfloor u\rfloor, z_{10}, z_{11}) \\
N_4(s,t,u,v,L_F)) &= N_1(v-\lfloor v\rfloor, z_{12}, z_{11})
\end{aligned}$$

3.2.1.1 Linear Interpolation Rolloff Correction

Rolloff error creates vignetting around an image, which is a darkening of pixels near the corners of an image due to the impossibility of an infinite sinc filter in real implementation. In the Fourier domain, the ideal filter is band-limited, but a real, practical filter falls away from the peak. This rolloff error can be compensated for by multiplying the plenoptic image by the reciprocal of the filter's inverse Fourier spectrum, which is also called the apodization function, to compensate for the darkening appearance due to the rolloff error [34].

The inverse fft of bilinear interpolation is a $\mathrm{sinc}^2(i, j)$ function. Equation (3.15) shows the premultiplication function, $A_l(i, j)$, used for correcting rolloff error

$$A_l(i,j) = (\frac{\sin c^2}{c^2}\frac{\sin r^2}{r^2})^{-1} \tag{3.15}$$

where m is the number of rows and n is the number of columns. The r and c terms are provided below in Equations (3.16-3.17).

$$r = \frac{i - \frac{m}{2}}{d} \tag{3.16}$$

$$c = \frac{j - \frac{n}{2}}{d} \tag{3.17}$$

where d is half of the diagonal of the image.

$$d = \frac{\sqrt{m^2 + n^2}}{2} \tag{3.18}$$

Figure 3.5 shows the resulting refocused image from rendering the premultiplied image using the FFT-based approach. The original rendered image itself does not have significant rolloff error, so the corrected image has a lightening around the edges for the rolloff corrected image.

(a) Rolloff correction (b) No rolloff correction

Figure 3.5: Rolloff correction for quadrilinear interpolation without oversampling

Figure 3.6b shows the extracted $m \times n$ image from an oversampled $2m \times 2n$ image using CUDA through the FFT-based rofocusing method, as described in Section 3.4.2.3.2. Vignetting is seen in this image due to oversampling in the Fourier domain to extract a larger Fourier slice to reduce aliasing. With rolloff correction in this case, the vignetting is significantly reduced. Compare the reduced aliasing around the clock in Figure 3.6 with the aliasing in Figure 3.5.

(a) Rolloff correction

(b) No rolloff correction

Figure 3.6: Rolloff correction for quadrilinear interpolation with oversampling

3.2.2 Kaiser-Bessel Interpolation

Fourier-Mellon interpolation is a higher order interpolation method used in generating plenoptic images to reduce aliasing [1]. For this dissertation, the 4-D Kaiser-Bessel interpolation follows the same structure as the 4-D quadrilinear interpolation to collapse the 4-D values through a series of 1-D interpolations, as seen in Equation (3.14), but with the 1-D Kaiser-Bessel interpolation equation, N_{KB}, rather than the linear interpolation equation seen in Equation (3.10). The 1-D Kaiser-Bessel interpolation is as follows:

$$N_{KB} = w_1 a_1 + w_2 a_2 \tag{3.19}$$

where

$$\begin{aligned} w_1 &= \frac{I_0(\beta\sqrt{1 - \frac{4(\frac{Mx}{2})^2}{(M-1)^2}})}{I_0(\beta)} \\ w_2 &= \frac{I_0(\beta\sqrt{1 - \frac{4(\frac{M(1-x)}{2})^2}{(M-1)^2}})}{I_0(\beta)} \end{aligned} \tag{3.20}$$

where β was chosen to be 3, M was chosen to be 500, and $I_0(n)$ is the zeroeth-order modified Bessel function of the first kind as shown below.

$$I_0(n) = \sum_{i=0}^{i_{max}} \frac{(\frac{x}{2})^{2i}}{i!^2} \tag{3.21}$$

The $I_0(\beta)$ term is precomputed since β is constant. a_1 and a_2 are the image or light field values at the corner coordinates.

3.2.2.1 Kaiser-Bessel Interpolation Rolloff Correction

The rolloff correction using Kaiser-Bessel interpolation is similar to the linear case; the goal is to reduce the effect of vignetting in the rendered images. The approximate inverse transform of the Kaiser-Bessel function, as discovered by Kaiser, is given in Equation (3.22) [35].

$$A_{KB}(i,j) = \frac{1}{k_x}\frac{1}{k_y} \tag{3.22}$$

where

$$k_x = \frac{1}{P_c}(M-1)\frac{\sinh(\pi\sqrt{(\frac{\beta}{\pi})^2 - (M-1)^2 f_r^2})}{I_0\pi\sqrt{(\frac{\beta}{\pi})^2 - (M-1)^2 f_r^2}} \tag{3.23}$$

$$k_y = \frac{1}{P_c}(M-1)\frac{\sinh(\pi\sqrt{(\frac{\beta}{\pi})^2 - (M-1)^2 f_c^2})}{I_0\pi\sqrt{(\frac{\beta}{\pi})^2 - (M-1)^2 f_c^2}} \tag{3.24}$$

and

$$f_r = 2N(\frac{i-\frac{m}{2}}{d}) \tag{3.25}$$

$$f_c = 2N(\frac{i-\frac{n}{2}}{d}) \tag{3.26}$$

where $P_c = 341.4$ is the peak of the curve used to normalize the window, d is the diagonal of the image, m is the number of rows, and n is the number of columns. Additionally, $2N = 0.004$ was chosen for the bin size, M was chosen to be 500, β was chosen to be 3, and $I_0(n)$ is the zeroeth-order modified Bessel function of the first kind as given in Equation (3.21). Note that separability is assumed for this equation; the 2-D implementation of the inverse transform was extended from the 1-D Kaiser-Bessel inverse transform.

Figure 3.7 shows the rolloff correction implementation using Equation (3.22). The rolloff corrected images between the linear interpolation of Figure 3.5 and the Kaiser-Bessel interpolation of Figure 3.7 shows a similarity for handling rolloff vignetting but with the Kaiser-Bessel implementation resulting in a qualitatively reduced compensation. Like the linear rolloff correction results, however, the rolloff error in the original images is not significant enough to warrant rolloff correction unless oversampling is used.

(a) Rolloff Correction (b) No rolloff correction

Figure 3.7: Rolloff correction for Kaiser-Bessel interpolation without oversampling

(a) Rolloff Correction (b) No rolloff correction

Figure 3.8: Rolloff correction for Kaiser-Bessel interpolation with oversampling

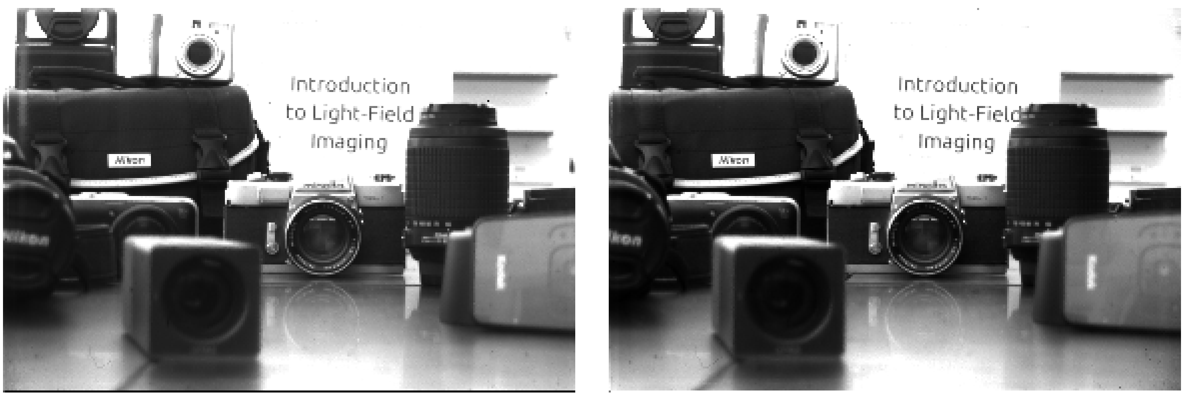
Oversampling using Kaiser-Bessel interpolation greatly increases the vignetting in the rendered image, as seen in Figure 3.8b. Rolloff correction, however, reduces this vignetting. Clearly, rolloff correction should only be implemented on images that suffer from vignetting.

3.2.3 A 2-D representation of the 4-D Radiance

Images of the 4-D radiance (including in the Fourier domain) are shown throughout this dissertation. This representation is a reordering of the four dimensions into two dimensions. Specifically, the $m \times n \times o \times p$ radiance image is shown as an array of images with dimensions $mo \times np$. The original plenoptic image is created in this fashion; however, the 2-D representation assumes a plaid structure and the indexing may be reordered, such as $om \times pn$. For complex terms (specifically for the Fourier domain), the magnitude is taken to incorporate both the real and imaginary parts of the complex number.

3.3 Perspective Imaging

Images rendered through perspective imaging uses each microlens to contribute one pixel to the rendered image at a chosen (u, v). Since the lightfield contains angular light rays, the perspective can be changed to simulate an angular change of the camera from a single image. Perspective images are quickly calculated by interpolating a point on the microlens for each pixel in the perspective image. Figure 3.9 shows two horizontal perspective images. Notice how much more the block obscures the camera in the center of the $(4, 8.5)$ image than the $(12, 8.5)$ image. Similarly, the vertical perspective images of Figure 3.10 show a significant difference



(a) Perspective image $(4, 8.5)$ (b) Perspective image $(12, 8.5)$

Figure 3.9: Horizontal perspectives

between the location of the near block from the bottom of the $(8.5, 4)$ image compared with the $(8.5, 12)$ image. Each pixel of the rendered $s \times t$ image I is calculated through a four

(a) Perspective image $(8.5, 4)$ (b) Perspective image $(8.5, 12)$

Figure 3.10: Vertical perspectives

dimensional interpolation on the light field as shown in Equation (3.27)

$$I(s,t) = N_4(s,t,u_d,v_d) \tag{3.27}$$

where N_4 is 4-D interpolation and u_d and v_d are the desired angular perspective. For integer microlens coordinates (u, v), no interpolation is needed, and each pixel can be pulled directly from the light field.

$$I(s,t) = L_F(s,t,u_d,v_d) \tag{3.28}$$

Since the perspective image is only a portion of the light field, a faster algorithm has been developed to render a perspective image using the raw plenoptic image itself and interpolates only the needed light field points for each pixel in the perspective image. This algorithm is described in Section 4.1.2.2.

3.4 Refocused Imaging

While the perspective image calculation renders an image based on a desired perspective, the plenoptic camera can also render images that simulate different camera focal lengths. Ng described two methods for rendering focal images: an integral-based method and a Fourier transform-based method [1].

3.4.1 Integral Refocusing Method

The raw plenoptic image consists of an $s \times t$ array of microlens, where each microlens consists of a $u \times v$ circular view through the microlens. Equation (3.29) describes the irradiance striking the sensor array [36].

$$E_F(s,t) = \frac{1}{F^2} \int \int L_F(s', t', u, v) \cos^4 \theta \, du \, dv \tag{3.29}$$

By reparameterizing the variables to a virtual film plane and absorbing the optical vignetting term, $\cos^4 \theta$, into the definition of the light field, the equation for the 4-D shear of the light field from a new refocused depth is given below [1].

$$L_F(s,t,u,v) = L_F(u(1 - \frac{1}{\alpha}) + \frac{s}{\alpha}, v(1 - \frac{1}{\alpha}) + \frac{t}{\alpha}, u, v) \tag{3.30}$$

Taking this shear and applying it to Equation (3.29) becomes the equation for integral refocusing, which can be thought of as shearing the light field and projecting it down into 2-D.

$$E_\alpha(s,t) = \frac{1}{\alpha^2 F^2} \int \int L_F(u(1 - \frac{1}{\alpha}) + \frac{s}{\alpha}, v(1 - \frac{1}{\alpha}) + \frac{t}{\alpha}, u, v) \, du \, dv \tag{3.31}$$

Therefore, the integral refocusing method sums the light rays which pass through the location of the simulated main lens location to generate the grayscale value at each pixel in the resulting $s \times t$ refocused image. In discrete terms and through the interpolation nomenclature of Section 3.2.1, Equation (3.31) becomes Equation (3.32)

$$I_{IR}(s,t) = \sum_{u=0}^{w_m} \sum_{v=0}^{h_m} N_4(-s', t', u, v) * A(u, v) \tag{3.32}$$

where (s', t') are as follows in Equations (3.33-3.34).

$$s' = u\alpha - \frac{w_m \alpha}{2} - s - u + \frac{w_m}{2} \tag{3.33}$$

$$t' = -v\alpha + \frac{h_m \alpha}{2} + t + v - \frac{h_m}{2} \tag{3.34}$$

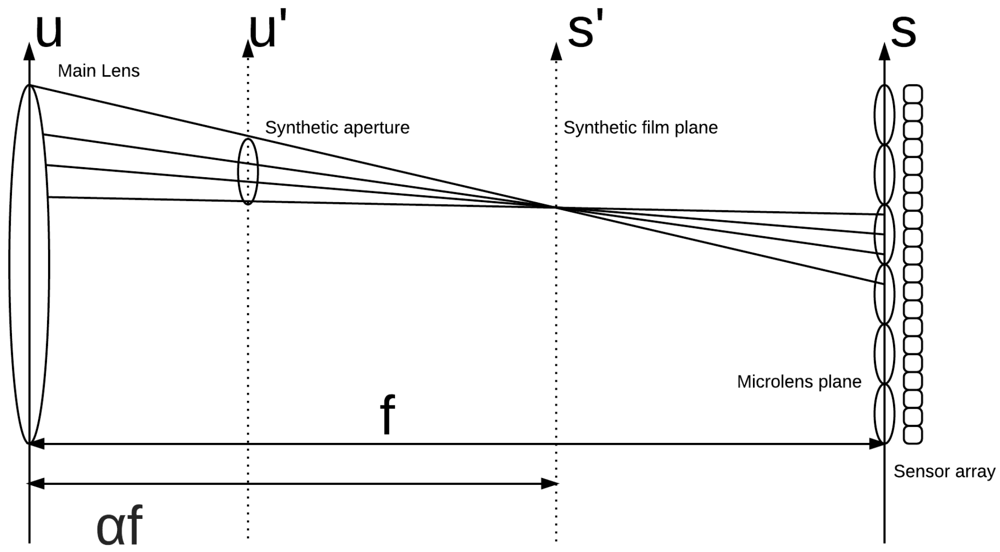


Figure 3.11: Refocused image diagram

In these equations, (s, t) are the pixel coordinates for the refocused image, (u, v) are the coordinates iterated over for integration, h_m is the height of the microlens, w_M is the width of the microlens, α is a scale factor of the focal length, f, that determines the focal depth, and $A(u, v)$ is the synthetic aperture for the simulated camera. The synthetic aperture for each (u, v) coordinate is either a 1 that represents open or a 0 that represents closed. Of particular note for Equation (3.32) is the transformation of a coordinate system with the origin in the corner (u, v, s, t) to a coordinate system where the origin is in the middle of image by the subtraction of half of the number of rows and columns.

(a) $\alpha = 2.0$

(b) $\alpha = 1.1$

Figure 3.12: Integral refocusing

(a) Aperture open (b) Aperture closed

Figure 3.13: Effect of aperture $A(u, v)$ on refocused image for $\alpha = 1.5$

3.4.1.1 Perspective Imaging with the Integral Method through Aperture

The aperture model $A(u, v)$ represents the aperture of the simulated light field. Since the integral method consists of the sum of light rays striking portions of the image, narrowing the aperture of the camera results in fewer light rays at each pixel in the image included in the summation. Biologically, this approach to sharpen a blurry view is to squint the eyes. The effect is an image with very wide depth of field, as seen in Figure 3.13. However, due to the increased computation required for the integral method, where most of the addends are multiplied by 0, the more efficient means of obtaining a full focus image is through the perspective imaging algorithm of Section 3.3.

3.4.2 Fourier Transform Refocusing Method

The Fourier transform refocusing method uses the Fourier slice theorem [37] to divide the Fourier transform of a 4-D light field image into a 2-D slice in the frequency domain. After taking the inverse 2-D Fourier transform, the resulting image is the refocused image.

3.4.2.1 The Fourier Transform

The Fourier transform is one of the most powerful and often used transforms in signal processing [38]. Its uses include medical imaging [39], GPS signal processing [40], and many others. In this dissertation, the Fourier Transform is used to render refocused images through the Fourier Slice Theorem, which is described in Section 3.4.2.2.1. This section describes the 2-D Fourier transform and its extension to four dimensions as used in this dissertation.

3.4.2.1.1 2-D Fourier Transform

The 2-D discrete Fourier transform, as it applies to images, is as follows:

$$F(\mathcal{S},\mathcal{T}) = \sum_{s=0}^{m-1}\sum_{t=0}^{n-1} f(s,t)e^{-j2\pi(\frac{\mathcal{S}s}{m}+\frac{\mathcal{T}t}{n})} \tag{3.35}$$

where $f(s,t)$ is an image of size $m \times n$. The inverse discrete Fourier transform is similar to Equation (3.35) except with a change in sign in the exponent.

$$f(s,t) = \frac{1}{mn}\sum_{\mathcal{S}=0}^{m-1}\sum_{=0}^{n-1} F(\mathcal{S},\mathcal{T})e^{j2\pi(\frac{\mathcal{S}s}{m}+\frac{\mathcal{T}t}{n})} \tag{3.36}$$

The $\frac{1}{MN}$ term is important, as different FFT implementations implement the scaled result (such as FFTW, which is described later) while others do not (such as the FFTW implementation in MATLAB). The 2-D FFT computational complexity is $O(n^2 \log n)$, or alternatively $O(4m^2 \log n)$.

3.4.2.1.2 4-D Fourier Transform

The 4-D discrete Fourier transform is an extension of the 2-D Fourier as follows:

$$F(\mathcal{S},\mathcal{T},\mathcal{U},\mathcal{V}) = \sum_{s=0}^{m-1}\sum_{t=0}^{n-1}\sum_{u=0}^{o-1}\sum_{v=0}^{p-1} f(s,t,u,v)e^{-j2\pi(\frac{\mathcal{S}s}{m}+\frac{\mathcal{T}t}{n}+\frac{\mathcal{U}u}{o}+\frac{\mathcal{V}V}{p})} \tag{3.37}$$

The inverse 4-D Fourier transform follows similarly to the 4-D Fourier transform but is not used or needed in this dissertation.

3.4.2.1.3 The Fast Fourier Transform

The fast Fourier transform (FFT) is a discrete Fourier transform algorithm that computes a Fourier transform in $O(n\log(n))$ time rather than $O(n^2)$. It first came into popularity through Cooley and Tukey in 1965 [41]. The Danielson-Lanczos Lemma divides the calculation of a

Fourier transform into smaller parts, which forms the basis of the reduction of the computational complexity to $O(n\log(n))$ time [42]. The discrete Fourier transform is expressed as the sum of two discrete Fourier transforms made from even-numbered indices and odd-numbered indices which is shown below for the 1-D case.

$$F(\mathcal{S})_s = \sum_{s=0}^{m-1} f(s)_n e^{-j2\pi(\frac{\mathcal{S}s}{m})} = \sum_{s=0}^{\frac{m}{2}-1} f(s)_{2s} e^{-j2\pi(\frac{\mathcal{S}2s}{m})} + \sum_{s=0}^{\frac{m}{2}-1} f(s)_{2s+1} e^{-j2\pi(\frac{\mathcal{S}(2s+1)}{m})} \tag{3.38}$$

The exponent term in the second term can be pulled out of the summation, which results in the same exponent computation in both summations.

$$F(\mathcal{S})_s = \sum_{s=0}^{\frac{m}{2}-1} f(s)_{2s} e^{-2j\pi\frac{s\mathcal{S}}{m/2}} + e^{-2j\pi\frac{\mathcal{S}}{m}} \sum_{s=0}^{\frac{m}{2}-1} f(s)_{2s+1} e^{-2j\pi\frac{s\mathcal{S}}{m/2}} \tag{3.39}$$

where $f(s)_{2s}$ are the even terms and $f(s)_{2s+1}$ are the odd terms. The exponent term outside of the summation term, $e^{-2j\pi\frac{\mathcal{S}}{m}}$, is called the twiddle factor [43], and can be precomputed for additional computational gain. This process of dividing up the indices can be repeated, and for m sizes that are a power of two, the computation is broken up into $\log(m)$ transforms of length 1, which is a classic ”divide and conquer” approach to get computational complexities of $O(n\log n)$ [44]. For m sizes that are not a power of two, the transform can be performed on indices corresponding to prime factors of m, which results in greater computational time. This particular trait of the FFT is further explored in Section 4.3.4.5. The Good-Thomas algorithm is an FFT algorithm that divides the m problem into sizes $m_1 m_2$ only if m_1 and m_2 are relatively prime [45]. Another important discrete Fourier transform for sizes of $m =$ 2,3,4,5,7,8,11,13 and 16 is the Winograd transform, which greatly increases the speed of the discrete Fourier transform [46].

3.4.2.2 Fourier Slice Theorem

The Fourier slice theorem is used to extract the 4-D Fourier transform of the radiance into a 2-D Fourier transform of the refocused image. Its origins lie in medical imagery [37]. The

2-D Fourier transform of the refocused image is the 2-D projection of the Fourier transform of the light field.

3.4.2.2.1 1-D Fourier Slice Theorem

In medical image processing, projections are an important aspect of computing images for computed tomography such as single photon emission tomograph (SPECT), positron emission tomography (PET), and magnetic resonance imagery (MRI). These projections are made up of lines from the medical images. A point in the projection is defined by the raysum along a line that is parameterized by Equation (3.40).

$$x \cos\theta_k + y \sin\theta_k = \rho_j \tag{3.40}$$

The raysum is defined as the sum of the rays along the line

$$g(\rho, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(x \cos\theta_k + y \sin\theta_k - \rho_j) dx\, dy \tag{3.41}$$

where g is parameterized in polar coordinates (ρ, θ) and (x, y) are in standard image coordinates. The δ operator requires its argument to be zero for any non-zero value. Equation (3.41) is the projection of $f(x, y)$ along a line in the $x - y$ plane and is called the Radon transform.

Taking the Fourier transform of the Radon transform with respect to ρ results in Equation (3.42).

$$\begin{aligned} G(\omega, \theta) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(x \cos\theta_k + y \sin\theta_k - \rho_j) e^{-j2\pi\omega\rho} dx\, dy\, d\rho \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \left[\int_{-\infty}^{\infty} \delta(x \cos\theta_k + y \sin\theta_k - \rho_j) d\rho \right] dx\, dy \end{aligned} \tag{3.42}$$

Using the impulse property of Equation (3.43) eliminates an integral.

$$\int_{-\infty}^{\infty} \delta(r(x, y, \theta)) f(\rho) d\rho = f(r(x, y, \theta)) \tag{3.43}$$

The resulting 1-D Fourier transform of a projection is then

$$\begin{aligned} G(\omega,\theta) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi\omega(x\cos\theta+y\sin\theta)}dx\,dy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}dx\,dy \end{aligned} \tag{3.44}$$

where $u = \omega\cos\theta$ and $v = \omega\sin\theta$. This expression is the Fourier transform of $f(x,y)$ at u and v.

$$\begin{aligned} G(\omega,\theta) &= F(u,v)|_{u=\omega\cos\theta, v=\omega\sin\theta} \\ &= F(\omega\cos\theta, \omega\sin\theta) \end{aligned} \tag{3.45}$$

The above equation makes up the Fourier slice theorem in which the Fourier transform of a projection is a slice of the 2-D Fourier transform of the region of the projection [37].

3.4.2.2.2 2-D Fourier Slice Theorem

This section extends Section 3.4.2.2.1 to the slice of a 4-D radiance in the frequency domain and for the context of plenoptic image refocusing. Ng generalized the Fourier Slice theorem to extend into higher dimensions through a change of basis [1]. The generalized Fourier slice theorem states that the Fourier transform of a change of the basis of an N-dimensional function f and an integral-projection to M of its dimensions is equivalent to taking the Fourier transform of f, changing the basis with the normalized inverse transpose of the original basis, and slicing it down to M dimensions [1].

$$\mathcal{F} \circ I_M^N \circ \beta = S_M^N \circ \frac{\beta^{-}T}{|\beta^{-}T|} \circ \mathcal{F}^N \tag{3.46}$$

This change of basis is followed by integral projection of the light field as follows in Equation (3.47)

$$I_2[L_F] = \frac{1}{\alpha^2 F^2} I_2^4 \circ \beta_\alpha[L_F] \tag{3.47}$$

where I_2^4 is the canonical projection operator that reduces a 4-D function down to 2-D by integration out the last 2 dimensions, β_{alpha} is an operator for an arbitrary change of basis of a

4-D function, α is the proportion of the main lens for refocusing, and L_F is the light field. This imaging change of basis is given in Equation (3.48).

$$\beta_\alpha = \begin{bmatrix} \alpha & 0 & 1-\alpha & 0 \\ 0 & \alpha & 0 & 1-\alpha \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.48}$$

The refocused imaging equation is then

$$I_{2\alpha} = \frac{1}{\alpha^2 F^2} \mathcal{F}^{-2} \circ (\mathcal{F}^2 \circ I_2^4 \circ \beta_\alpha) \tag{3.49}$$

where $\circ$ is the composition of functions operator. Using the generalized Fourier slice theorem of Equation (3.46), results in the following equation.

$$I_{2\alpha} = \frac{1}{\alpha^2 F^2} \mathcal{F}^{-2} \circ (S_2^4 \circ \frac{\beta_\alpha^{-T}}{|\beta_\alpha^{-T}|} \circ \mathcal{F}^4) \tag{3.50}$$

Since $|\beta_\alpha^{-T}| = |\beta_\alpha^{-1}| = \frac{1}{\alpha^2}$,

$$I_{2\alpha} = \frac{1}{F^2} \mathcal{F}^{-2} \circ S_2^4 \circ \beta_\alpha^{-T} \circ \mathcal{F}^4 \tag{3.51}$$

Equation (3.51) is the final equation for calculating a refocused image from a 4-D radiance with the FFT refocusing method. The Fourier transform of the 4-D radiance, $\mathcal{F}$, undergoes a change of basis, β_α^{-T}, then a slice from 4-D to 2-D, S_2^4, and finally an inverse 2-D Fourier transform, $\mathcal{F}^{-2}$ followed by a scale $\frac{1}{F^2}$.

The Fourier Slice Photograph Theorem states that the footprint of all full-aperture 2-D photographs lies on the following 3-D manifolds in the 4-D Fourier space [1]

$$(\alpha k_x, \alpha k_y, (1-\alpha) k_x, (1-\alpha) k_y) \tag{3.52}$$

where $\alpha \in [0, \infty)$ and $k_x, k_y \in \mathbb{R}$. Note that the α term only extends from $[0, \infty)$. A focal stack of all depths, then, is not equivalent to the full 4-D light field [1]. The extent that the α

term only covers positive values is an important distinction between the FFT-based refocusing method and the integral-based refocusing method.

3.4.2.3 Aliasing Concerns

Aliasing is always present in digital images, since an image has only finite dimensions [37]. Spatial aliasing arises due to the lack of sufficient sampling at a rate lower than the Nyquist rate. The two major aliasing problems in plenoptic imaging are rolloff artifacts [47], which result in a darkening of the rendered image borders, and post-aliasing artifacts [48], which are phantom artifacts of features in an image [1]. Of greatest concern for this dissertation is the post-aliasing artifacts due to possible erroneous feature detection on the phantom artifacts. Ng cites three methods for suppressing post-aliasing artifacts [1].

- Oversampling
- Superior filtering
- Zero-Padding

The first two methods, oversampling and superior filtering, are described in the next sections, while zero-padding in CUDA is described in Section 4.3.4.5.

3.4.2.3.1 Superior filtering with Kaiser Bessel Interpolation

Quadrilinear interpolation is a simple and fast interpolation algorithm but is susceptible to aliasing. Its implementation in refocusing with the FFT is useful for creating the 4-D radiance, since using Kaiser-Bessel interpolation for calculating the radiance does not seem to improve the image rendering quality significantly. Kaiser-Bessel interpolation, which is described in Section 3.2.2, reduces aliasing due to its superior filter characteristics. Figure 3.14 shows two images rendered from a plenoptic image of a sign placed .305 meters from the camera. The left image using quadrilinear interpolation shows the aliasing as phantom letters, which are more easily seen in dark regions due to their lower energy. Using Kaiser-Bessel interpolation on the image on the right, the aliasing is significantly reduced.

(a) Aliasing with quadrilinear interpolation

(b) Reduced aliasing with Kaiser-Bessel interpolation

Figure 3.14: Effect of interpolation on aliasing at $\alpha = 1.13$

Computational Times for Kaiser-Bessel Interpolation

The computational cost of calculating the Kaiser-Bessel interpolation function is largely dependent on the max iterations i_max for the zeroeth-order modified Bessel function. Table 3.1 shows the computation time required for the Fourier slice, which uses Kaiser-Bessel interpolation on the 4-D radiance.

Table 3.1: Fourier Slice with Kaiser-Bessel Interpolation at $\alpha = 1.03$

i_{max}	1	2	3	4	5	6	7
Fourier Slice (ms)	0.8539	9.2470	47.1421	59.8325	73.5071	87.3233	101.4594

With insufficient iterations, the image quality suffers greatly. A i_max of The Kaiser-Bessel window can also be optimized further for greater computational efficiency [49]. Figure 3.15 shows the maximum iterations of 1,2,7, and 15 for the zeroeth-order modified Bessel function. A single iteration results in very poor image quality. With the introduction of one additional iteration, the image quality improves significantly. With seven iteration, some of the interpolation artifacts, as seen at the top of the image, are decreased. At fifteen iterations, the computation time increases to 219.8 milliseconds, and the image is almost indistinguishable from the $i_{max} = 7$ case.

(a) $i_{max} = 1$

(b) $i_{max} = 2$

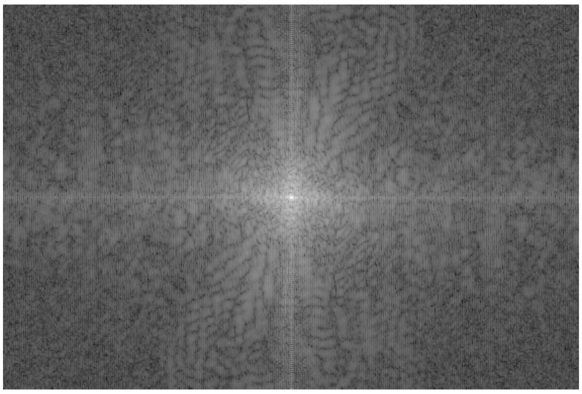
(c) $i_{max} = 7$

(d) $i_{max} = 15$

Figure 3.15: Effect of various maximum iterations at $\alpha = 1.03$

3.4.2.3.2 Oversampling in the Fourier Domain

The four dimensional Fourier transform of the light field can be interpolated into higher resolution images through oversampling. Figure 3.16a shows an interpolated 2-D slice that is oversampled by two, resulting in a resolution that is two times larger than the usual case for the Fourier slice. As mentioned in [1], increasing the sampling rate in the Fourier domain increases the replication period in the spatial domain, which results in less energy in the replicas that fall within the rendered image. On taking the inverse 2-D FFT, the refocused image as well as some aliases can be seen in Figure 3.16b. Finally, cropping and normalizing the refocused image results in the final refocused image seen in Figure 3.16c.



(a) Resampled (2x) Fourier slice

(b) Inverse 2-D FFT for the refocused image

(c) Cropped and normalized refocused image

Figure 3.16: Oversampling and its effect on the Fourier slice

Chapter 4

Plenoptic Image Real-time Rendering

One of the biggest hurdles to any localization algorithm is developing software that is capable of real-time processing. The image processing required for plenoptic imaging is a significantly difficult problem due to the raw plenoptic image size. For example, the processing time required for computing the dense depth map in [50] was more than 1000 seconds. In another example by Dansereau, the processing time using MATLAB code was about a half second per frame [51]. The algorithms described in Chapter 3 can be implemented across many different platforms and languages. For this dissertation, the programming language was chosen to be C++, with parallel processing implemented through a multicore CPU implementation known as OpenMP [52] and a GPU implementation known as CUDA [53].

This chapter is divided into three sections covering single-threaded, multi-threaded, and GPU-based processing of the image rendering methods. Each section contains a description and timing of the radiance generation, perspective imaging, integral-based refocusing, and the FFT-based refocusing algorithms and any improvements to increase the speed of the algorithm. The final section describes each pipeline for generating the images and the fastest times for each of them as well as a discussion on the merits of each algorithm for the purposes of generating point clouds. Section 4.1 in this chapter describes the platform independent improvements to the perspective imaging and two refocused image methods. Section 4.2 describes the OpenMP implementation for a multi-threaded implementation of each image rendering method. Finally, Section 4.3 presents a short explanation of CUDA, some problems associated with using a GPU implementation, and the CUDA-based implementation of the perspective and two image refocusing methods.

4.0.2.3.3 Execution Times

The execution times presented in this section were measured using the `gettimeofday` function available on Unix platforms or the CUDA event available for measuring time as seen in Code Listings 4.1 and 4.2. While the `gettimeofday` function is affected by processes that effect the system clock, such as an Network Time Protocol server [54] or a change to the system time by the user, the short times required for processing the rendered images makes the `gettimeofday` function sufficient for measuring time to a millisecond [55].

Listing 4.1: Measuring Time with gettimeofday

```
gettimeofday(&tickT,0);
//function_to_measure();
gettimeofday(&tockT,0);
double startTime=static_cast<double>(tickT.tv_sec)+static_cast<double>(tickT.tv_usec)*1e-6;
double endTime =static_cast<double>(tockT.tv_sec)+static_cast<double>(tockT.tv_usec)*1e-6;
// function_to_measure took (endTime - startTime) seconds to run
```

Listing 4.2: Measuring Time with CUDA Events

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start);
//function_to_measure();
checkCuda(cudaEventRecord(stop));
checkCuda(cudaEventSynchronize(stop));
checkCuda(cudaEventElapsedTime(&milliseconds, start, stop));
// function_to_measure() took milliseconds to run
```

All tests were conducted in Ubuntu version 14.04 on an Intel Core i7-4720HQ CPU with 4 cores (8 with multithreading), a GeForce GTX 980M graphics card, a 1 TB spinning disk hard drive, and 16 gigabytes of Random Access Memory (RAM). The number of other running processes was minimized. Because the operating system used in these timings is not a real-time operating system, each time measurement was repeated at least 1000 times to obtain the average time for execution.

4.0.2.3.4 CUDA Execution Time Pitfalls

The addition of a separate piece of hardware can result in false measurement times. One easy pitfall is not checking for CUDA errors, which can result in execution times that are significantly better than reality, since the asynchronous call to the GPU abruptly ends on errors and control is returned to the CPU. One example of this problem is when data pointers reference the same place on GPU memory as previous code executions. When the kernel call fails and returns control to the host code, the execution time is faster, and the data transport pulls data from an address on the GPU that already contains the processed data. The resulting rendered image appears correct, and computation time is significantly improved despite the failed kernel execution. This problem was mitigated through the selection of a random plenoptic image to ensure that the rendered image was not the result of transporting old data from the GPU.

4.1 Single Core Execution

Single core execution of the image rendering algorithms generally follow the algorithms described in Chapter 3. The algorithm's implementation is described in this section. Some additional enhancements to speed up processing are also described. The datatype used for the data structures is 32-bit `float`. However, the raw plenoptic images are 16-bit, 3280 $\times$ 4904 images. Care must be taken when converting the data between different bit depths. The rendering images have a resolution of $m \times n$, or 286 $\times$ 190, as either 16-bit or 8-bit images which are downsampled from the 32-bit data used for the interim datatypes. Appendix A shows the execution tables for the 64-bit `double` precision as well as the CUDA datatype `cufftDoubleComplex`. Single precision, 32-bit data of type `float` is significantly faster and is described in the following sections.

4.1.1 Radiance Generation

The radiance generation implementation consists of four nested `for` loops that iterate through each pixel in a $m \times n$ radiance image that interpolates the raw plenoptic image using the microlens centers to achieve a plaid radiance image. The radiance can be one of two types depending on the image rendering method. For the integral refocusing or the perspective imaging, the standard radiance is created. The FFT refocusing method requires a shift of the zero frequency to the center of the image rather than at the edges. This shift consists of either multiplying each pixel by negative number or doing nothing at all, depending on the index. The shift results in additional processing and is discussed further in Section 4.1.4.2.

The corners of each of the microlens images do not need interpolation since they have no light striking the sensor. As such, any interpolation for these pixel values are a waste of computational power. Any (u, v) coordinates that are within four pixels from the corners are set to 0 in the radiance image, as shown in Equation (4.1).

$$L_F = \begin{cases} 0 \text{ for } u < 4 \wedge v < 4 \\ 0 \text{ for } u > 12 \wedge v > 12 \\ 0 \text{ for } u < 4 \wedge v > 12 \\ 0 \text{ for } u > 12 \wedge v < 4 \\ N_2(c_x(s,t) - r + u, c_y(s,t) - r + v, I) \text{ otherwise} \end{cases} \tag{4.1}$$

where $\wedge$ is the logical OR function. Note that for the standard (non-shifted) version of the radiance generation, these additional logic checks are slower than the 2-D interpolation.

4.1.1.1 Serial Radiance Generation Execution Time

Table 4.1 shows the execution time for serial processing of both the shifted and non-shifted radiance. Using a single thread, processing the radiance data structure takes almost a quarter of a second. Since the radiance is generally the first step in rendering an image from the plenoptic camera, the serial case is not sufficient for operating in real-time. The difference between the shifted and non-shifted versions of generating results in a sub-millisecond difference.

Table 4.1: Serial Times for Radiance Generation

Radiance Timing	**(ms)**
Non-shifted	248.5014
Shifted	248.3686

4.1.2 Perspective Imaging

The perspective image rendering process iterates through each pixel of the rendered image of size $m \times n$ and interpolates the desired (u, v) perspective over either the radiance or the original plenoptic image. Unlike the original plenoptic image, the radiance is plaid in a regular grid and has already been compensated for subpixel coordinates. Therefore, additional computation is needed for processing over the original plenoptic image to ensure the resulting rendered perspective image is generated from the plaid (s, t, u, v) coordinates.

4.1.2.1 Perspective Image from Radiance

The serial perspective imaging algorithm for calculating the perspective image from the radiance relies on one major algorithmic improvement: removing the interpolation from the algorithm when the desired coordinates are integers. In this case, the interpolation has no effect on the output. By ignoring the interpolation code completely and obtaining the perspective point directly from the radiance, computation times are significantly reduced. In the general case, the serial algorithm for perspective imaging iterates through each pixel in the $m \times n$ image and calculates the subpixel point in the radiance through 4-D interpolation.

4.1.2.2 Perspective Image from Plenoptic Image

Like the perspective imaging algorithm from the radiance, the primary increase in computational efficiency comes from ignoring the interpolation when the desired (u, v) coordinates are integers. If the coordinates are not integers, this increased efficiency is lost. For calculating the perspective image from the plenoptic image directly, plaid coordinates must be calculated before either calculating the 4-D interpolation for the perspective image pixels or directly recording the pixels (in the case of integer coordinates). The plaid coordinate calculation consists of a 2-D interpolation over the desired coordinates needed by the 4-D interpolation.

The basic improvement of the perspective imaging from the plenoptic image over the perspective imaging from the radiance is caused by not needing to calculate the radiance. On a $m \times n$ image, or a 190×286 image, the perspective imaging algorithm requires 54,340 4-D interpolation operations. For the perspective from plenoptic approach, each of these 4-D interpolation operations requires an additional 16 2-D interpolation operations to generate the plaid coordinates. However, the perspective from radiance approach processed an additional $m \times n \times o \times p$, or 190×286×16×16 2-D interpolation operations to generate the radiance, which is then followed by the aforementioned 4-D interpolation operations. Perspective imaging from the radiance is only beneficial when the radiance has already been generated for a particular plenoptic image.

4.1.3 Integral Refocused Imaging

The serial integral refocusing image equations were described in Section 3.4.1. The practical implementation of these equations consists of four nested `for` loops. The outer two loops iterate over the rendered $m \times n$ image, while the inner two loops consist of the summation of Equation (3.32) over the $o \times p$ elemental images. Code Listing 4.3 shows the nested `for` loops used for calculating the integral refocused image, including the summation variable `sum`.

Listing 4.3: Integral Refocused Loops

```
for(sPrime = 0; sPrime < size1; sPrime++) {
  for(tPrime = 0; tPrime < size2; tPrime++) {
      sum = 0;
      for(uPrime = 0; uPrime < size3; uPrime=uPrime+1) {
         for(vPrime=0; vPrime < size4; vPrime = vPrime+1) {
```

In the above code, the terms `sPrime`, `tPrime`, `uPrime`, and `vPrime` represent s', t', u', v', respectively, and `size1`, `size2`, `size3`, and `size4` represent m, n, o, p, respectively. For the innermost loop, the (u, v) coordinate frame must shift the origin to the center of the elemental image.

$$\begin{aligned} u_n &= u - \frac{o}{2} \\ v_n &= -(v - \frac{p}{2}) \end{aligned} \tag{4.2}$$

The u'' and v'' variables are merely the original (u, v) coordinates. The s'' and t'' variables are then calculated using the shifted (u_n, v_n) coordinates.

$$\begin{aligned} s'' &= -(u_n + \frac{-s' - u_n}{\alpha})\alpha \\ t'' &= (v_n + \frac{t' - v_n}{\alpha})\alpha \end{aligned} \tag{4.3}$$

With the newly calculated (s'', t'', u'', v'') coordinates, the summand σ can be determined from the 4-D interpolation of the radiance and the aperture A as shown below.

$$\sigma_{s,t,u,v} = N_4(s'', t'', u'', v'', L_F)A(u'', v'') \tag{4.4}$$

When (s'', t'', u'', v'') are whole number integers, Equation (4.4) can be replaced for an efficiency gain by Equation (4.5) because no interpolation is needed.

$$\sigma_{s,t,u,v} = L_F(s'', t'', u'', v'')A(u'', v'') \tag{4.5}$$

In Equations (4.4) and (4.5), the aperture A is set to either `1` for open or `0` for closed. For the perspective integral refocusing mentioned in Section 3.4.1.1, the aperture A is closed except for the perspective of interest (u, v), where $A(u, v)$ represents the desired perspective. The summand is then added to the final summation ε.

$$\varepsilon = \varepsilon + \sigma_{s,t,u,v} \tag{4.6}$$

For all σ values that are not out of the bounds of the image, a variable γ representing the count, or total number, of valid sums is incremented. The final rendered pixel at coordinate (s', t') is then provided below.

$$I(s', t') = \frac{\varepsilon}{\gamma} \tag{4.7}$$

This division of the sum is used to reduce vignetting by normalizing the intensity of neighboring pixels [1].

4.1.3.0.1 Region of Interest Implementation

Rendering a small portion of the image can be a desirable trait for some algorithms, such as feature point tracking where the only relevant portions of the refocused image is around the feature points. Since the two outer loops iterate over the rendered refocused image, these loops can be reduced down to refocus only a portion of the image. A `cv::Rect` object is used for declaring the desired region of interest for the newly refocused image, which consists of an (s, t) coordinate for the initial point with a $w_r \times h_r$ side length for the box. The refocused image with the reduced region of interest is therefore a $w_r \times h_r$ image. Code Listing 4.4 shows the code required to implement the region of interest implementation of the integral refocused method. Instead of iterating over the width and height, or $m \times n$ of the rendered image, I, the region of interest integral refocused image iterates from a given start point over a given width, $xSize$, and height, $ySize$.

Listing 4.4: Integral Refocused Loops - Region of Interest

```
for(sPrime = xSearch; sPrime < xSize; sPrime=sPrime+1) {
  for(tPrime = ySearch; tPrime < ySize; tPrime=tPrime+1) {
```

In addition to the change over the (s', t') iteration, the coordinates must be modified on the smaller $w_r \times h_r$ refocused rendered image. Equation (4.8) shows the summed value ε and the division of the number of detected light rays γ to reduce vignetting.

$$I(s' - x_s, t' - y_s) = \frac{\varepsilon}{\gamma} \tag{4.8}$$

where y_s corresponds to the `ySearch` variable and x_s corresponds to the `xSearch` variable. This equation converts the full rendered image coordinates (s', t') to (s'_{roi}, t'_{roi}).

4.1.4 FFT Refocused Imaging

The FFT refocused method uses the Fourier slice method to generate refocused images. As described in Section 3.4.2, the FFT Refocusing method consists of four primary steps:

- Shifted Radiance Generation

- 4-D FFT of Shifted Radiance
- 2-D Slice
- 2-D iFFT of Slice

The following sections will describe each of these steps. However, first some important distinctions between the analytical FFT used for generating the equations of Section 3.4.2 and the FFTW implementation is given.

4.1.4.1 FFTW

One of the most common implementations of the Fourier transform uses the Fastest Fourier Transform in the West (FFTW) [56]. This dissertation uses FFTW as the implementation of the FFT on the CPU.

4.1.4.1.1 FFTW Plans

FFTW creates a `fftw_plan` which consists of the most efficient means of calculating a particular FFT given the size of the data, the dimensionality of the data, and the direction (direct or inverse). Code Listing 4.5 shows the initialization of the FFTW plan for both the 4-D FFT of the radiance, `fftRadiancePlan`, as well as the 2-D inverse FFT of the Fourier slice, `ifftImage`. Typically, this process takes a significant amount of time for the FFTW library to determine the best FFT algorithm for this particular implementation. This plan is calculated once and then saved to reduce code initialization time. Note that the FFTW plan does not need to be calculated in real-time.

Listing 4.5: Integral Refocused Loops - Region of Interest

```
int n[4] = {points.size1(), points.size2(), microLensWidth, microLensHeight};
int n2[2] = {points.size2(), points.size1()};
fftw_complex* dataIn = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
    n[0]*n[1]*n[2]*n[3]);
fftw_complex* out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * n[0]*n[1]*n[2]*n[3]);
fftw_complex* out2 = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * n[0]*n[1]);
fftw_complex* dataOut2 = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * n[0]*n[1]);
fftRadiancePlan = fftw_plan_dft(4,n,dataIn,out,FFTW_FORWARD,FFTW_MEASURE);
```

```
ifftImagePlan = fftw_plan_dft(2,n2,out2,dataOut2,FFTW_BACKWARD,FFTW_MEASURE);
```

Several important distinctions exist between the analytical FFT, the C FFTW implementation used in this dissertation and the implementation of FFTW used in MATLAB. The next section describes these differences.

4.1.4.1.2 Analytical Fourier Transform vs. FFTW Fourier Transform

A crucial part of understanding a practical implementation of the FFT-based refocusing algorithm is the difference between the analytical Fourier transform and the transform calculated through the Fastest Fourier Transform in the West (FFTW) [56]. FFTW is a widely-used implementation of the Fourier transform that is included both as a separate library as well as used in MATLAB [56]. While the algorithm described in Section 3.4.2 assumes an analytical implementation of the FFT, FFTW implements a shifted and scaled version of the FFT.

Scaling between MATLAB FFTW vs. C library FFTW

One difference between MATLAB's FFTW implementation and the C FFTW implementation is that MATLAB's FFTW implementation is automatically scaled by the problem size, while the C FFTW implementation is not. This distinction must be made when comparing the two versions of FFTW.

Shifting between Analytical FFT and FFTW

To demonstrate the difference between the analytical FFT and MATLAB's FFT, an analytical analysis of the 2-D FFT of a Gaussian curve is presented below. The equation for a 2-D Gaussian curve is given in Equation (4.9).

$$f(x,y) = Ae^{-(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2})} \tag{4.9}$$

For simplicity's sake, assume $\sigma_x = \sigma_y = \sigma$ and $x_0 = y_0 = 0$.

$$f(x,y) = Ae^{-(\frac{(x)^2}{2\sigma^2}+\frac{(y)^2}{2\sigma^2})} \tag{4.10}$$

A 2-D function is separable if it can be written as the product of two functions of a single variable. Through some manipulation, Equation (4.10) can be simplified.

$$f(x,y) = \sqrt{A}e^{\frac{-x^2}{2\sigma^2}}\sqrt{A}e^{\frac{-y^2}{2\sigma^2}} \tag{4.11}$$

Through separability, Equation (4.11) can be described as

$$f_{xy}(x,y) = f_x(x)f_y(y) \tag{4.12}$$

where

$$f_x(x) = \sqrt{A}e^{\frac{-x^2}{2\sigma^2}} \tag{4.13}$$

$$f_y(x) = \sqrt{A}e^{\frac{-y^2}{2\sigma^2}} \tag{4.14}$$

The Fourier transform of a 2-D function, then, is the product of two 1-D functions.

$$\mathscr{F}\{f(x,y)\}(k) = \mathscr{F}\{f_x(x)\}\mathscr{F}\{f_y(y)\} \tag{4.15}$$

The analytical fourier transform is given in Equation (4.16).

$$X(f) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi ft}dx \tag{4.16}$$

Taking the FFT of the first factor begins as shown in Equation (4.17).

$$\mathscr{F}\{f(x)\}(k) = \mathscr{F}\{\sqrt{A}e^{\frac{-x^2}{2\sigma^2}}\} \tag{4.17}$$

$$\mathscr{F}\{f(x)\}(k) = \sqrt{A}\int_{\infty}^{\infty} e^{\frac{-x^2}{2\sigma^2}}e^{-2\pi jkx}dx \tag{4.18}$$

Using Euler's formula,

$$e^{ix} = \cos x + j\sin x \tag{4.19}$$

Equation (4.18) substituted using Euler's formula is given below.

$$\mathscr{F}\{f(x)\}(k) = \int_{-\infty}^{\infty} e^{\frac{-x^2}{2\sigma^2}} [\cos(2\pi kx) - j\sin(2\pi kx)]dx \tag{4.20}$$

Separating the integral into two integrals yields the following equation.

$$\mathscr{F}\{f(x)\}(k) = \int_{-\infty}^{\infty} e^{\frac{-x^2}{2\sigma^2}} \cos(2\pi kx)dx - j\int_{-\infty}^{\infty} e^{\frac{-x^2}{2\sigma^2}} \sin(2\pi kx)dx \tag{4.21}$$

The second term in Equation (4.21) is odd, and as such goes to 0.

$$j\int_{-\infty}^{\infty} e^{\frac{-x^2}{2\sigma^2}} \sin(2\pi kx)dx = 0 \tag{4.22}$$

The first integral is given in [57].

$$\int_{-\infty}^{\infty} e^{\frac{-x^2}{2\sigma^2}} \cos(2\pi kx)dx = \sqrt{2\pi A\sigma^2} e^{-2\pi^2\sigma^2k^2} \tag{4.23}$$

The analytical FFT for the x variable is therefore

$$\mathscr{F}\{f(x)\}(k) = \sqrt{2\pi A\sigma^2} e^{-2\pi^2\sigma^2k^2} \tag{4.24}$$

Proceeding similarly for the y variable results in Equation (4.25).

$$\mathscr{F}\{f(y)\}(n) = \sqrt{2\pi A\sigma^2} e^{-2\pi^2\sigma^2n^2} \tag{4.25}$$

The equation for the analytical FFT of a 2-D exponential function is therefore

$$\mathscr{F}\{f(x,y)\}(n,k) = 2\pi A\sigma^2 e^{-2\pi^2\sigma^2(k^2+n^2)} \tag{4.26}$$

Figure 4.1 shows the analytical FFT of Equation (4.26) on the left plot. Since the analytical solution assumes that the zero frequency occurs at the image center, the resulting FFT has the zero frequency at the image center. FFTW (in this case implemented in MATLAB) assumes that the zero frequency is at image coordinates $(1, 1)$, or the top left corner of the image, and so the resulting FFT is shifted, which results in the image seen in the right side of Figure 4.1. This offset will result in a failed rendering solution for the Fourier slice.

(a) Analytical 2-D FFT (b) FFTW FFT-no shift

Figure 4.1: Analytical FFT vs. FFTW for $A = 5$ and $\sigma = .01$: no shift

Figure 4.2 shows the resulting FFT calculated by FFTW when the shift is included. In this case, the analytical and computed FFT are the same within a very small tolerance (.002 pixels with no zero padding). The next section describes how the shift was conducted in 4-D for the shifted radiance.

(a) Analytical 2-D FFT (b) FFTW FFT-shift

Figure 4.2: Analytical FFT vs. FFTW for $A = 5$ and $\sigma = .01$: shift

4.1.4.2 Shifted Radiance Generation

The first step in creating an FFT-based refocused image is generation of a plaid radiance. Just as in the other rendered images, the radiance is a 4-D data structure with (s, t, u, v) as the main coordinate system. Prior to taking the FFT of the radiance, however, the origin of the coordinate frame must be shifted from the edges of the image to the center of the image. For a 2-D image, this process is simply a reorganization of the quadrants of the image. For a 4-D image, the conceptual idea of shifting a 4-D data structure is much more difficult. Fortunately, a convenient property of the FFT will handle the shift automatically.

$$\mathscr{F}(\mathcal{S} - \frac{m}{2}, \mathcal{T} - \frac{n}{2}, \mathcal{U} - \frac{o}{2}, \mathcal{V} - \frac{p}{2}) \iff f(s,t,u,v)(-1)^{s+t+u+v} \tag{4.27}$$

where $\mathcal{S}$, $\mathcal{T}$, $\mathcal{U}$, $\mathcal{V}$ are the 4-D indices in the Fourier domain. Any shift in the Fourier domain, then, can be conducted by multiplying the 4-D function by -1 to the sum of the (s,t,u,v) indices. As a matter of fact, a shift in any number of dimensions can be conducted by multiplying -1 to the sum of the indices of each coordinate.

The shift operation requires an iteration through the 4-D data structure. Unfortunately, the FFTW library does not allow for an iteration through the radiance when computing the FFT. The only time a full 4-D iteration is conducted on the radiance is during its creation. To eliminate the need for an additional iteration through the 4-D data structure, this shifting operation will be done during the generation of the radiance, since the negative term flows through the FFT algorithm like any other scalar. The modification to the radiance generation step is as follows:

$$L_F = \begin{cases} N_2(c_x(s,t) - r + u, c_y(s,t) - r + v, I) \text{ for } (s+t+u+v) \bmod 2 \equiv 0 \\ -N_2(c_x(s,t) - r + u, c_y(s,t) - r + v, I) \text{ otherwise} \end{cases} \tag{4.28}$$

Note that Equation (4.28) does not explicitly calculate $(-1)^{s+t+u+v}$, since the exponential function is significantly less efficient than the modulo operator. Instead, the shift implementation only checks if the summation of the indices is even or odd through the use of the modulo operator. Even sums result in no change to the radiance term, while odd sums result in the negative of the radiance term. The effect on performance of this additional operation can be seen in Section 4.2.1.

4.1.4.2.1 Shifted Radiance and cuFFT

For cuFFT, the CUDA FFT implementation discussed in Section 4.3.4, the shift can occur during the FFT algorithm explicitly because the implementation of the transpose requires an iteration through the entire radiance image, which opens up the capability of applying the negative then. For consistency with the FFTW method, however, the CUDA algorithm assumes an already shifted radiance.

4.1.4.3 4-D FFT of Shifted Radiance

The implementation of the 4-D FFT of the shifted radiance on the CPU is straightforward, as seen in Code Listing 4.6. Using the 4-D FFT plan developed in Section 4.1.4.1, the FFT is calculated. The inputs to the `fftw_execute_dft` function are the plan, the input `fftw_complex* radiance` array, and the output `fftw_complex* fftRadiance` array in the Fourier domain.

Listing 4.6: FFT Execution using FFTW

```
fftw_execute_dft(plan,radiance,fftRadiance);
```

The implementation of the 4-D FFT is more difficult on the GPU due to the limitations of the CUDA FFT implementation. Section 4.3.4.2 describes of the 4-D FFT on the GPU.

4.1.4.4 2-D Fourier Slice

The Fourier slice implementation consists of an iteration through the $m \times n$ rendered Fourier slice, as shown in Code Listing 4.7. Each (k_x, k_y) coordinate is referenced from the shifted Fourier domain, and the iteration proceeds under the assumption that the origin is in the center of the 4-D radiance image.

Listing 4.7: FFT Execution using FFTW

```
for(ky = -size2/2; ky < size2/2; ky = ky + 1)
{
  for(kx = -size1/2; kx < size1/2; kx = kx + 1)
   {
```

A new set of coordinates, $(\mathcal{S}, \mathcal{T}, \mathcal{U}, \mathcal{V})$, are calculated as the slice of the 4-D radiance in the Fourier domain.

$$
\begin{aligned}
\mathcal{S} &= k_x \\
\mathcal{T} &= k_y \\
\mathcal{U} &= (\alpha - 1)k_x \\
\mathcal{V} &= (\alpha - 1)k_y
\end{aligned}
\tag{4.29}
$$

where α is the depth discriminating refocused variable, and $(\mathcal{S}, \mathcal{T}, \mathcal{U}, \mathcal{V})$ are the coordinates of the underlying 2-D Fourier slice.

Like many of the other rendering algorithms, the ability to remove the interpolation from the algorithm is a significant advantage for computational efficiency. For the 2-D slicing algorithm, the interpolation is unneeded if $\alpha = 1$. In this case, each pixel in the Fourier slice is present in the 4-D Fourier radiance, and the interpolation is ignored. For any other α values, a 4-D interpolation is done using the $(\mathcal{S}, \mathcal{T}, \mathcal{U}, \mathcal{V})$ coordinates on the 4-D radiance in the Fourier domain. As a point of concern, the indexing for the data structure must be shifted to account for the (k_x, k_y) shifted coordinate frame. In addition, the Fourier domain itself must be shifted to move the origin back to the edges of the image. This operation, unlike the 4-D shift, only requires shifting over the (k_x, k_y) coordinate frame as seen in Equation (4.30)

$$
\mathcal{I}(k_x + \frac{m}{2}, k_y + \frac{n}{2}) = \begin{cases} N_4(\mathcal{S}, \mathcal{T}, \mathcal{U}, \mathcal{V}, \mathcal{L}_F) \text{ for } (k_x + k_y) \bmod 2 \equiv 0 \\ -N_4(\mathcal{S}, \mathcal{T}, \mathcal{U}, \mathcal{V}, \mathcal{L}_F)) \text{ otherwise} \end{cases}
\tag{4.30}
$$

where $\mathcal{L}_F$ is the radiance in the Fourier domain.

Notice that Equation (4.29) is the first reference to the depth discrimination variable α in the FFT-based refocused image algorithm. Since each slice, or depth image, is dependent on the 4-D FFT of the radiance, but the α term is only used after the 4-D FFT has been calculated, an entire stack of refocused images can be quickly calculated from the 4-D FFT of the radiance through an $m \times n$ image. The capability of rendering different refocused depths without the need to iterate through the entire $m \times n \times o \times p$ 4-D radiance is the greatest advantage of the FFT-based refocused imaging algorithm.

4.1.4.5 2-D iFFT of Fourier Slice

The implementation of the 2-D inverse FFT of the Fourier slice is, like the 4-D FFT of the shifted radiance, straightforward and is shown in Code Listing 4.8. Using the 2-D inverse FFT plan developed in Section 4.1.4.1, the inverse 2-D FFT is calculated for the Fourier slice. The parameter listings to the `fftw_execute_dft` function are once again the plan, the input `fftw_complex*` slice array, and the output `fftw_complex*` image array in the Fourier domain.

Listing 4.8: Inverse FFT Execution using FFTW

```
fftw_execute_dft(plan,slice,image);
```

4.1.4.6 Weak Edge Detection

A weak edge image can be detected from the shifted radiance by implementing the 1-D interpolation algorithm N_1 using half integer (u, v) coordinates such as $(8.5, 8.5)$. This operation is similar to an edge image kernel with -1 and 1. Equation (4.31) shows the kernel used to generate the simple edge detector of Figure 4.3a.

$$K_{edge} = \begin{bmatrix} -1 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.31}$$

This kernel was chosen to approximate the weak edge detector seen in Figure 4.3b instead of other kernels that would give a better edge map. Note that the edge detector extracted from the shifted radiance has almost completely missing edges near the nominal focus of the camera such as the markings of “USA” and “IMPERX”. This loss of very clear edges is not desired for an edge detector but could be useful for determining features that are beyond the nominal focus distance of the plenoptic camera.

(a) Simple edge detector

(b) Edge image from extracting perspective image from shifted radiance using $(8.5, 8.5)$

Figure 4.3: Weak edge detection through perspective imaging

4.1.5 Serial Imaging Execution Times

The execution times for running the aforementioned algorithms with a single thread is shown in Table 4.2. For the calculation of a single perspective image, the rendering from the raw plenoptic image is best. The majority of the computation time required for perspective imaging from the radiance image is generating the radiance itself. This increased computation time propagates to both the integral and FFT refocused rendering algorithms as well.

Table 4.2: Serial Times for Image Rendering

Algorithm	**(ms)**	
	General	*Integer*
Perspective from Radiance	254.1	249.2
Perspective from Plenoptic	21.8	3.4
Integral Refocused	699.1	196.5
FFT Refocused	717.3	706.4

4.2 OpenMP

Generation of the plenoptic imagery, like many image processing algorithms, is a highly parallel operation. There are several application program interfaces (API) available for utilizing multiple cores of a modern microprocessor. These include POSIX threads, or pthreads [58], Message Passing Interface (MPI) [59], and Cilk [60]. OpenMP [52] was chosen as the platform for multithreaded programming and provides developers with a means to create shared memory, multi-threaded parallel applications [52].

One common use of OpenMP is to wrap the for loops with OpenMP directives to automatically parallelize the for loop. Code Listing 4.9 shows the OpenMP loop construct for parallelizing a for loop. Each for loop runs on a thread and continues until the end of the OpenMP block or another OpenMP directive.

Listing 4.9: OpenMP Schedule Directive

```
#pragma omp for schedule(dynamic,chunk) nowait
```

The #pragma omp directive describes the start of the worksharing region, which is the section of code that is parallel. The schedule divides each iteration in the for loop into size chunk. With an eight core (assuming a hyperthreading [61]) microprocessor, the chunk size has been chosen to be $\frac{n}{c_n}$, where $c_n = 8$ is the number of cores used in the computation and n is the amount of data to be divided among the threads. The dynamic protocol distributes threads as needed to the available cores. The nowait clause prevents the threads from waiting for all threads before proceeding past the worksharing region.

The code Listing below shows the variables used in the for construct. In this example, the image is shared across all threads, while the i,j variables are private to each thread similarly to their use in a serial for loop. Note that no explicit threading, semaphores, or mutexes are needed with OpenMP; the parallelization is done through the OpenMP directives automatically.

Listing 4.10: Directives for Shared Variables Across Threads

```
#pragma parallel shared(image) private(i,j)
```

4.2.1 Radiance Generation

The OpenMP implementation of the radiance generation is similar to the serial implementation, except with the addition of the OpenMP directives before the for loops as shown in Code Listing 4.11. The data structure holding the radiance image, radiance, is shared between all of the threads for storing each of the calculated values in the 4-D structure. The four indices, (s, t, u, v) for the 4-D radiance are private to each thread. The `collapse(n)` directive indicates that each thread should process the code within the n^{th} `for` loop, so for `collapse(2)`, each thread will be processing a $o \times p$ number of index elements for the radiance. For `collapse(4)`, each thread processes each individual index element for the radiance; however, no significant speedup in processing was found for `collapse(4)`. Since an $m \times n$ number of threads are needed to process the data over the two `for` loops, the `chunk` size was changed to $\frac{mn}{n_c}$.

Listing 4.11: Radiance Generation OpenMP directives

```
#pragma omp parallel shared(radiance) private(s,t,u,v)
   {
#pragma omp for schedule(dynamic,chunk) nowait collapse(2)
```

Table 4.3 shows the average processing time over 1000 iterations for the shifted (for the FFT refocusing) and non-shifted (for the integral refocusing and perspective radiance) methods. The additional computation required to implement the shift is not a significant computational burden, especially since an additional iteration through the entire 4-D radiance is not needed.

Table 4.3: Multicore Processing Time for Generating Radiance

Radiance Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Non-Shifted	253.3	154.7	121.9	100.3	86.4	72.1	62.9	55.3
Shifted	251.6	159.7	121.3	104.4	88.7	74.8	64.8	56.8

4.2.2 Perspective Imaging

The perspective imaging method, like the radiance generation, requires only minor changes to apply multithreaded capability with OpenMP. Code Listing 4.12 shows the OpenMP directives added to the serial code. The image is shared among all threads, and the indices (i, j) are private to each thread. The `collapse(2)` directive is used here so that each thread computes one interpolation for the pixel rather than iterating over the inner `for` loop. Like the radiance generation, the `chunk` size is $\frac{mn}{n_c}$.

Listing 4.12: Perspective Imaging OpenMP Directives

```
#pragma omp parallel shared(image) private(i,j)
{
#pragma omp for schedule(dynamic,chunk) nowait collapse(2)
```

4.2.2.1 OpenMP Perspective Imaging Total Execution Times

Figure 4.4 shows the block diagram of the steps for calculating the perspective image from the radiance as well as the general case processing times for the serial and OpenMP cases. The number in parenthesis is the speedup of the OpenMP processing time over the serial time. The radiance generation is a significant portion of processing the perspective image, since the perspective image rendering time is a small fraction of the total computation times.

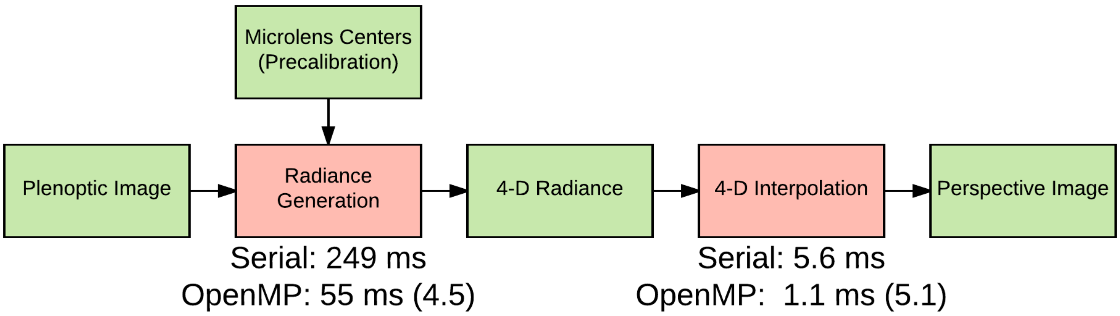


Figure 4.4: Perspective imaging from the radiance

Figure 4.5 shows the perspective rendering steps for calculating the perspective image directly from the plenoptic image. Since no radiance generation is required, the processing time is significantly reduced. Notice, however, that the time to render the perspective image from the plenoptic image (5.6 ms) is longer than the time to generate the perspective image from the radiance (1.1 ms) since the perspective image from plenoptic time requires additional interpolation steps. Table 4.4 and Table 4.5 show the computation time for rendering the perspective image from the radiance and the plenoptic image as an average over 1000 runs, respectively. As the number of cores increase, the computation time generally drops. However, the advantage of multiple cores quickly reaches diminishing returns, and the computation for each thread

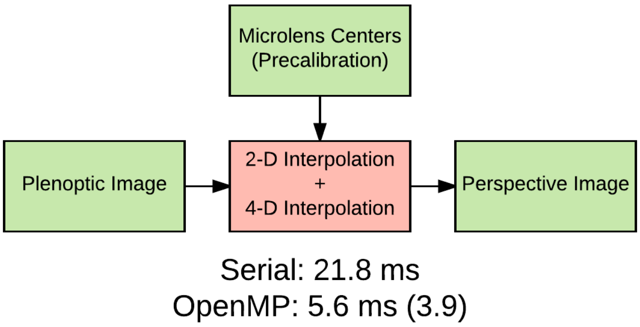


Figure 4.5: Perspective imaging from the plenoptic image

Table 4.4: Multicore Processing Time for Generating Perspective Image from Radiance

Perspective Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
General	5.3903	3.4522	2.5181	1.9983	1.5883	1.3550	1.1893	1.0897
Integer	2.0476	1.3157	1.0013	0.8819	0.7529	0.7286	0.7135	0.6667

is not significant enough to easily make up for the overhead associated with setting up multiple threads. Rendering a perspective image from the radiance or the plenoptic image is not a computationally heavy operation. The additional computation time required by calculating the image from the plenoptic image shows that calculating the plaid coordinates prior to the 4-D interpolation is a significant factor in its computational efficiency.

Table 4.5: Multicore Processing Time for Generating Perspective Image from Plenoptic Image

Perspective Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
General	20.3	15.8	10.9	8.4	6.9	5.9	5.7	5.6
Integer	5.3	2.8	2.0	1.7	1.6	1.5	1.3	1.3

However, the need to calculate the radiance quickly shows the advantage of directly computing the perspective image from the plenoptic image. Table 4.6 shows the total execution times needed for the general and the integer cases. The radiance generation is a significant increase in execution time, and so the perspective imaging from the plenoptic image is always preferred except when the radiance has already been calculated. Note that the radiance created for Table 4.6 is the non-shifted version of the algorithm, since only the FFT algorithms require a shifted radiance.

Table 4.6: Total Multicore Processing Time for Generating Perspective Image from Radiance

Perspective Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Radiance	253.4	154.8	122.0	100.4	86.4	72.1	62.9	55.3
General	5.4	3.5	2.5	2.0	1.6	1.4	1.2	1.1
Total	258.8	158.3	124.5	102.4	87.9	73.5	64.1	56.4
Radiance	253.4	154.8	122.0	100.4	86.4	72.1	62.9	55.3
Integer	2.0	1.3	1.0	0.9	0.8	0.7	0.7	0.7
Total	255.4	156.1	123.0	101.2	87.1	72.8	63.6	56.0

4.2.3 Integral Refocused Imaging

The multithreaded integral refocusing algorithm adds the OpenMP directives to the integral refocused imaging algorithm. Code Listing 4.13 shows the additional OpenMP directives added directly before the `for` loops.

Listing 4.13: Integral Refocused Imaging OpenMP Directives

```
#pragma omp parallel shared(destImage)
private(sPrime,tPrime,vPrime,uPrime,sum,count)
{
#pragma omp for schedule(dynamic,chunk) nowait collapse(2)
```

Since any summation is not ideal for parallelization, the `collapse(2)` directive is used to wrap the outer `for` loops to parallelize the computation for the rendered image coordinates $I(s', t')$. The inner `for` loops are run by each thread to calculate the sum ε.

Table 4.7 shows the average execution times for float values (General) and integer values (Integer) of α over 1000 runs. For the single core general calculation, the execution time is greater than one second, which is extremely poor performance for a real-time application. The integer version of the refocused image, which avoids interpolation, runs at about a fifth of the general case, which shows the significant processing times required for interpolation. Like the other threaded algorithms, larger number of cores results in diminishing returns for execution speedup.

Table 4.7: Multicore Processing Time for Generating Integral Refocused Image

Refocused Timing	**(s)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
General	840.6	563.6	417.7	348.4	290.7	245.9	212.7	186.8
Integer	339.8	192.9	130.3	102.5	84.3	71.8	62.6	55.7

Region of Interest Refocusing Algorithm

For some applications, the entire refocused image is not needed; only a small portion of the image needs to be refocused (such as on a particular feature). The region of interest refocusing algorithm takes advantage of the scaling ability of the refocusing algorithm to reduce computation time of small subsets of the refocused image. Table 4.8 and Figure 4.6 show the parallel computation time with 8 threads required for calculating a small subset of the image. For this data, the *Scale* term refers to the $s \times tn$ size of the image subset, where n is the scale factor used in Table 4.8. Figure 4.6 shows the total number of pixels processed to generate the refocused region of interest. As the number of processed pixels increases, the required computation time increases linearly, as expected.

4.2.3.1 OpenMP Integral Refocused Imaging Total Execution Times

Figure 4.7 shows the integral refocused block diagram and its associated general case for the OpenMP processing times along with the serial case for comparison. Computation of the radiance is a significant factor in processing the integral refocused image, but the actual computation of the integral image is even longer. OpenMP provides speedups of 4.5 and 2.4 for the radiance generation and integral refocusing steps, respectively.

Tables 4.9 and 4.10 show the total computation time for the integral refocusing method using multiple cores. The single core case is effectively the serial imaging step with some

Table 4.8: Processing Time for Generating Refocused Image - Region of Interest

Timing	**(ms)**									
Scale	*.05*	*.10*	*1.5*	*.20*	*.25*	*.30*	*.35*	*.4*	*.45*	*.50*
Parallel	8.0	17.4	25.6	37.9	44.6	53.1	68.0	71.9	81.7	99.6
Parallel (Integer)	3.5	6.2	8.8	11.6	14.5	16.9	19.8	22.3	25.1	27.8
Serial	33.3	79.9	103.1	139.6	191.7	209.6	246.1	277.6	312.3	348.4
Serial (Integer)	9.2	19.8	28.8	39.2	49.2	58.9	69.6	78.7	88.7	98.2
Scale	*.55*	*.60*	*.65*	*.70*	*.75*	*.80*	*.85*	*.90*	*.95*	*1.0*
Parallel	100.3	109.9	118.7	138.7	138.0	146.5	172.0	164.8	174.3	183.0
Parallel (Integer)	30.4	33.1	35.8	39.5	42.3	44.8	47.6	50.2	52.9	55.6
Serial	381.0	417.3	450.1	492.3	529.7	564.3	596.2	628.1	672.0	706.0
Serial (Integer)	107.3	117.9	126.8	138.4	147.1	156.5	166.9	176.0	186.0	196.1

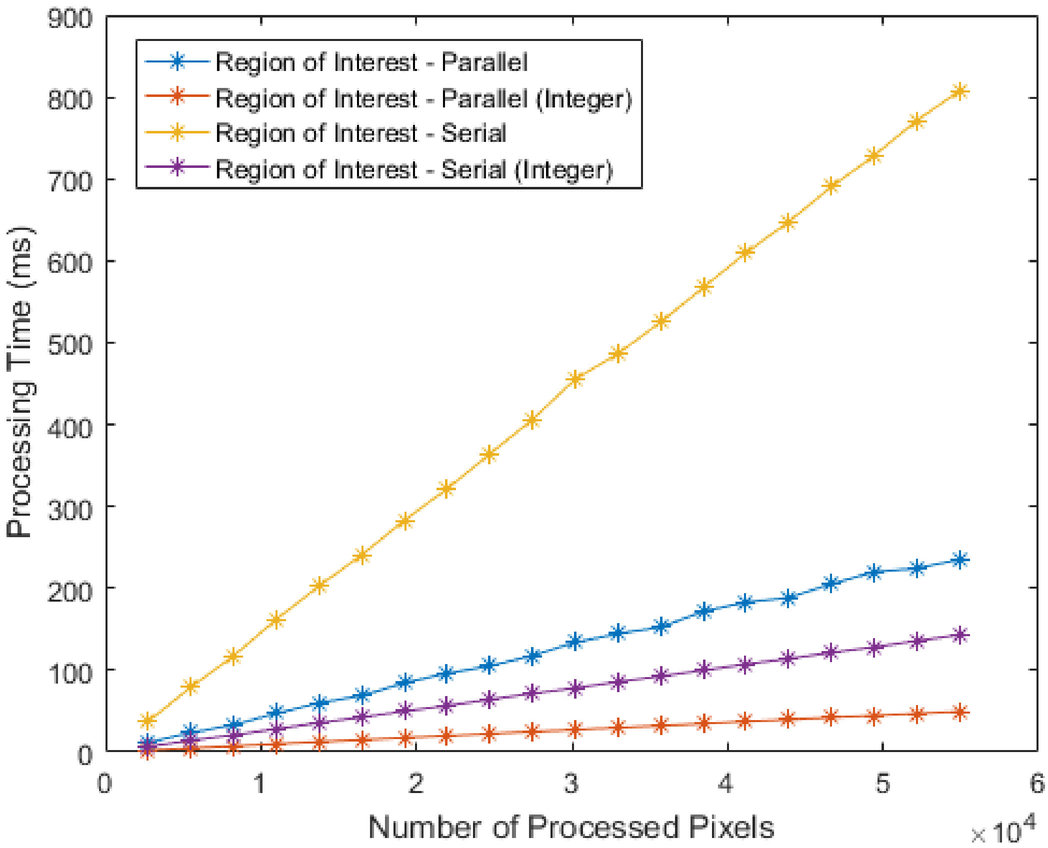


Figure 4.6: Integral refocus region of interest processing times

OpenMP overhead. The eight core case is the best case scenario for a multi-threaded application without additional hardware such as a GPU, and its execution time is almost 300 ms. Note that the speedup increase gets smaller with each additional core. This diminishing returns with

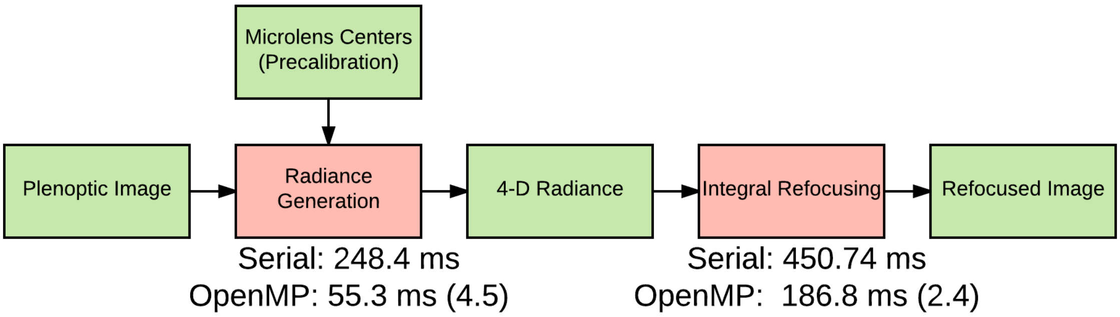


Figure 4.7: Integral refocusing

additional cores is a staple of parallel processing algorithms, since the job size for each core gets incrementally smaller as more cores are used. For example, each core in a two core system has half of the total job required, and so half of the total job is additionally parallelized. With 3 cores present, a third of the total job is assigned to each core, and only one sixth of the total job is additionally parallelized over the 2 core case. Diminishing returns at even the 8 core case, however, are still advantageous, resulting in an almost 50ms gain from 7 cores to 8 cores for the integral refocusing method.

Table 4.9: Multicore Processing Time for Generating Integral Refocused Image

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Non-Shifted Radiance	253.4	154.8	122.0	100.4	86.4	72.1	62.9	55.3
Integral Refocusing	840.6	563.6	417.7	348.4	290.7	245.9	212.7	186.8
Total	1094.0	718.4	539.7	448.8	377.0	318.1	275.6	242.2

The α integer case requires significantly less computation time with only about 110 ms required for execution. Unlike perspective imaging, the integer case is extremely limited in that only the $\alpha = 1$ case is realistically expected to be generated. Other integer values of α are far from the nominal focal length of the camera.

Table 4.10: Multicore Processing Time for Generating Integral Refocused Image - Integer

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Non-Shifted Radiance	253.4	154.8	122.0	100.4	86.4	72.1	62.9	55.3
Integral Refocusing	339.8	192.9	130.3	102.5	84.3	71.8	62.6	55.7
Total	593.2	347.7	252.3	202.9	170.6	143.9	125.5	111.0

4.2.4 FFT Refocused Imaging

Each of the four steps in the FFT Refocused imaging algorithm can be sped up by taking advantage of multithreaded capabilities. For the shifted radiance generation and 2-D Fourier slice algorithms, OpenMP directives were used. For the 4-D FFT and 2-D inverse FFT, however, the number of threads had to be specified with an additional line prior to creating the `fftw_plan`, as shown in Code Listing 4.14. By calling the `fftw_plan_with_nthreads` function, the number of threads, `numThreads`, can be specified. The execution times of Table 4.11 were determined through calculation of the `fftw_plan` where the thread numbers are as stated in the table.

Listing 4.14: FFTW Plan Creation - multiple threads

```
fftw_plan_with_nthreads(numThreads);
fftRadiancePlan = fftw_plan_dft(4,n,dataIn,out,FFTW_FORWARD,FFTW_MEASURE);
ifftImagePlan = fftw_plan_dft(2,n2,out2,dataOut2,FFTW_BACKWARD,FFTW_MEASURE);
```

The shifted radiance generation follows just as the non-shifted version as described in Section 4.2.1 with the exception of the additional code to handle the -1 scalar for odd sums of indices. The 2-D slice is similar to the perspective image through the use of the `collapse(2)` directive to wrap the two `for` loops as well as using the radiance as a shared variable and the two loop variables as private variables, as shown in Code Listing 4.15. The chunk size for the `dynamic` directive is $\frac{mn}{n_c}$.

Listing 4.15: OpenMP directives for Fourier Slice

```
#pragma omp parallel shared(radiance) private(kx,ky)
{
#pragma omp for schedule(dynamic,chunk) nowait collapse(2)
```

4.2.5 OpenMP FFT-based Refocused Imaging Total Execution Times

Figure 4.8 shows the OpenMP execution times for the general case for each step in the FFT-based refocused imaging algorithm along with the serial case for comparison. Generation of the shifted radiance was not significantly slower than the generation of the non-shifted radiance. The OpenMP calculation of the 4-D FFT improves over the serial time by almost a factor of 3. The computation times for the 4-D slice and the 2-D inverse FFT, however, is very short with times of 2.6 ms and 0.4 ms for the OpenMP case, respectively.

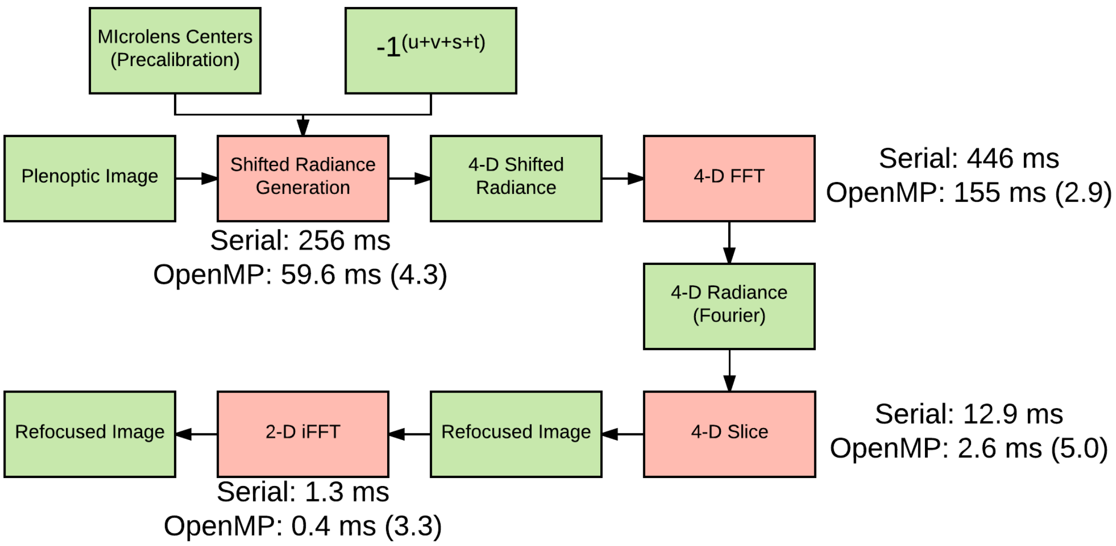


Figure 4.8: FFT refocusing

Tables 4.11 and 4.12 show the execution times for each of the four steps of the FFT-based refocusing algorithm. As expected, the calculation of the shifted radiance and its FFT make up the bulk of the required time to execute. The speedup from having no 4-D interpolation in the $\alpha = 1$ case is lost under the shifted radiance and 4-D FFT operations. Since the α term is only used in the 4-D slice, different refocused images from the same plenoptic image can be quickly recalculated using the 4-D Fourier domain radiance data construct. For applications that require a large focal stack for each plenoptic image, the FFT-based method has the clear advantage in processing time over the integral refocusing.

Table 4.11: Multicore Processing Time for Generating FFT Refocused Image

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Shifted Radiance	256.5	163.1	124.7	108.2	92.5	77.9	67.5	59.6
4-D FFT	446.6	249.1	264.2	173.0	202.7	171.8	158.5	154.8
4-D Slice	12.9	9.1	6.3	4.7	0.6	3.3	2.8	2.6
2-D iFFT	1.3	0.7	0.6	0.4	0.6	0.5	0.5	0.4
Total	717.3	422.0	395.8	286.3	296.5	253.5	229.3	217.4

Table 4.12: Multicore Processing Time for Generating FFT Refocused Image - $\alpha = 1$

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Shifted Radiance	256.0	160.2	123.3	109.3	92.3	77.9	67.5	59.9
4-D FFT	443.9	249.2	269.7	179.8	190.9	170.3	159.2	151.8
4-D Slice	5.2	2.5	1.6	1.2	1.1	0.9	0.8	0.7
2-D iFFT	1.3	0.7	1.0	0.4	0.6	0.5	0.5	0.4
Total	706.4	412.7	395.7	290.8	285.0	250.0	228.0	212.7

4.3 CUDA

While multithreaded capabilities can greatly decrease processing time, implementation of the image rendering algorithms on a Graphic Processing Unit (GPU) has even greater potential. GPU's are specialized processing units originally designed to rapidly calculate 3-D computer graphics. Development of additional capabilities of the GPU has led to the use of GPU's for machine learning [62], physics simulations [63], cryptography [64], and image processing. The GPU's inherent parallel processing capability makes it ideal for parallel operations. Nvidia has developed an application programming interface (API) called CUDA for its brand of GPU's [53]. This section describes the implementation of the generation of the radiance, perspective images, integral refocused images, and the FFT-based refocused images on a GPU.

CUDA, as implemented for image rendering in this dissertation, is built upon the asynchronous execution of a section of code on the GPU called a kernel. Kernels are executed on the GPU, which is called the device, while the CPU, called the host, calls the kernel code for execution. The host/device labels are not software-defined labels; data must be passed between the host and device and can be a significant bottleneck. The kernel is comparable to the code contained inside the `for` loops of the OpenMP algorithm. Each kernel executes in a thread, and up to 32 threads can execute concurrently in what is called a warp. Threads are organized according to blocks, with 1024 threads per block. Since each warp executes with 32 threads, the desirable thread size for each block is a multiple of 32.

Array Indexing

Since each thread corresponds to one execution of the kernel, each thread has unique variables for array indexing. Code Listing 4.16 shows the host code responsible for dictating the grid and thread sizes for processing the radiance.

Listing 4.16: Grid and Thread Deployment -Host

```
dim3 grid(size2, size1, 1);
dim3 threads(microLensWidth, microLensHeight, 1);
```

The `grid` variable is a three element vector made up of the sizes for each dimension of the CUDA grid construct. In this case, the grid is arranged as a $m \times n$ array. The `thread` variable describes the sizes of the CUDA thread construct, which is an $o \times p$, or 16×16 size array of threads. The combination of grid and thread blocks creates a 4-D block for processing or iterating through the radiance, where the thread blocks process the elemental image indices and the grid blocks iterate over the elemental images. Since the thread block is of size 16×16, the total number of threads is 256 threads, which adheres to the desired thread size of a multiple of the warp size. However, 256 is only a quarter of the maximum thread size of 1024. While from an intuitive standpoint maximizing the number of available threads would be a greater advantage than using only a fraction available, the need for data transfers and pipelining for all of the threads can reduce the speed at which the threads are run. An example is given in Section 4.18 for CUDA processing on a variety of thread block sizes where greater use of threads results in greater computational time.

Listing 4.17: 4-D Grid and Thread Deployment: Radiance Iteration on Device

```
t = blockIdx.x;
s = blockIdx.y;
v = threadIdx.x;
u = threadIdx.y;
```

Since the maximum number of dimensions for the thread size is three, the 4-D radiance is indexed using the indices of the CUDA blocks as seen in Code Listing 4.17. These variables are the same as those described in the serial algorithm description of Section 4.1 and the multithreaded implementation in Section 4.2. With this approach, any frequent data reads when processing the elemental images (typically for 4-D interpolation) are near each other in thread blocks. This approach is also intuitively easy to understand for mapping the 2-D grid and thread block sizes into 4-D. An alternative approach is filling up a 3-D block of threads for indexing to maximize block use, but the 1024 limit on thread size is too small for a $n \times o \times p$ data block.

However, the perspective images and the 4-D slice require a different approach, since both of these algorithms only iterate over two `for` loops instead of four. The radiance array indexing iterates over the $m \times n$ dimension through the grid sizes. This approach, when used with the perspective and 4-D slicing algorithms, would effectively use only a single thread from each grid. Instead, a more traditional CUDA indexing operation is used for these algorithms, as seen in Code Listings 4.18 and 4.19, where the half `size` variable for the 4-D slice is used prior to shifting the zero frequency out of the center. In both of these code listings, the `kx` and `ky` variables are the indices used in the data addresses of the 2-D rendered images. By incorporating the indices in this manner, more than one thread is used within each grid for greater computational power.

Listing 4.18: 2-D Grid and Thread Deployment: 4-D Slice on Device

```
int kx = (blockIdx.x*size1 + threadIdx.x) - size1/2;
int ky = (blockIdx.y*size2 + threadIdx.y) - size2/2;
```

Listing 4.19: 2-D Grid and Thread Deployment: 2-D Perspective Imaging on Device

```
int kx = (blockIdx.x*blockDim.x + threadIdx.x);
int ky = (blockIdx.y*blockDim.y + threadIdx.y);
```

Data Type Conversion

The standard compiler for the serial and the multi-threaded portion of the plenoptic image rendering code is the GNU compiler, or gcc [65]. The compiler for CUDA, nvcc, is not a full C++ compiler. To allow the standard compiler to compile the non-CUDA portions of the code, the CUDA portion of code is compiled separately into a library. This separable compilation results in a datatype conversion problem as data is passed between the CUDA-capable library and the main executable. If datatypes from one compiler are not bit-capable with datatypes from the other, a memory copy would need to be done to move data between the CUDA code and the serial and multi-threaded parts of the plenoptic imagery code. Fortunately, the CUDA capable code has parallels that directly allow variables to be cast back and forth. The list below describes these parallels.

- `std::complex<double>` and `cufftDoubleComplex`
- `std::complex<float>` and `cufftComplex`
- OpenCV's `Mat.data` and the data parameter for `cudaMemcpy`

The pointers to the data on the GPU are stored in a C++ class to allow the data on the GPU to remain in scope and accessible to any other code. It also allows the important data variables, such as the radiances and the images, to be allocated in the constructor and avoid that computational cost during image rendering.

4.3.1 Radiance Generation

As previously mentioned, any iteration through the radiance, such as for radiance generation, relies on the array indexing of Code Listings 4.16 and 4.17. The radiance generation kernel requires the raw plenoptic image, the output radiance data structure, and the calibration points of the center of the microlens. The radiance generation algorithm itself is similar to the algorithms for the serial and the multithreaded case. The calibration points for the center of the microlens are preloaded into the GPU to remove any transport time during image rendering. The radiance generation uses dynamic parallelism, which allows a kernel to call another kernel. For creating the plaid radiance data structure, dynamic parallelism is most often used in the 2-D interpolation used for obtaining subpixel coordinates. Dynamic parallelism is available for GPU's with compute capability greater than 3.5. The following sections describe the radiance generation for each algorithm. Additionally, the next section describes an important part of a CUDA implementation - the transport time cost.

4.3.1.1 CPU → GPU and GPU → CPU Data Transport

A significant factor in using CUDA is the transport time of data, since the GPU is additional hardware that requires data to be physically sent to it from the CPU. Two options are available for sending data to the GPU and back - pageable memory and pinned memory. Pageable memory is data allocations that are accessible by the CPU. When the GPU needs data, CUDA must first allocate pinned memory, copy the pageable memory to the pinned memory, and finally send the data to the GPU. The process can be shortened by allocating the memory on the CPU directly as page-locked, or pinned, memory using the `cudaMallocHost` function. The rate at which these data transfers occur is highly dependent on the system [66]. Table 4.13 shows the transport times for pageable and pinned memory. For this particular system, the pinned and pageable memory transfer times are not significantly different.

Table 4.13: CUDA Processing time for Pageable vs. Pinned Memory

Transport Timing	**(ms)**
Pageable Transport	4.338
Pinned Generation	3.058

4.3.1.2 Data Transfer Rate

The data transfer rate is the amount of data that can be sent from the host to the device, or vice versa, over a period of time. The plenoptic image has a data type of `unsigned char*`, or an unsigned 16-bit image. As an image with a height of 3280 pixels and a height of 4904 pixels, the total number of bytes that must be sent from the CPU to the GPU is 32,170,240. The data transfer rate, then is 7.29 GB/s for this system.

Table 4.14 shows a summary of the five main data structures in the CUDA pipeline. The 32 MB plenoptic image is significantly smaller than the radiance size, which is an $m \times n \times o \times p$, or 286 $\times$ 190 $\times$ 16 $\times$ 16, 4-D data structure of type `float`, or a 32-bit data structure. The total number of bytes for the integral-based radiance image is 111,288,320 bytes, which is far larger than the 16-bit plenoptic image. The data structure needed to hold the FFT of the radiance is even larger due to the complex numbers in the Fourier domain. The radiance used in the FFT-based refocusing is of type `cufftComplex`, which is two values of type `float` for the real and imaginary parts of the complex number, or twice the size of the integral radiance. The total number of bytes of the FFT-based radiance image is 111,288,320 bytes, or about three times the size of the raw plenoptic image. However, since both radiance types are only on the GPU, no additional time is needed to move the 32-bit radiance compared with the 16-bit plenoptic image.

Table 4.14: CUDA Data Structure Sizes

Data Structure Size	**(Bytes)**
Plenoptic Image	32,170,240
Radiance (Integral)	55,644,160
Radiance (FFT)	111,288,320
Image (Perspective and Integral)	217,360
Image (FFT)	434,720

Transport of the rendered image is another source of time delay for the CUDA system. The integral refocusing image and the perspective images are of 32-bit type `float`, while the FFT-based rofocused rendered image is of type `cufftComplex`, or a 64-bit image with real and imaginary components. The FFT-based refocused rendered image is more than four times the size of the integral and perspective images only due to the data type. Smaller data types are crucial for fast transport times. An additional kernel, shown in Code Listing 4.20, is used to move the data contained in the `cufftComplex` data structure to a `float` data type both for faster transport time and to express the image as the magnitude of its complex numbers.

Listing 4.20: Datatype Conversion

```
int kx = blockIdx.x*blockDim.x + threadIdx.x;
int ky = blockIdx.y*blockDim.y + threadIdx.y;
if(kx < size1 && ky < size2)
   destImage[(int)((kx)+size1*(ky))] =
       sqrt(diffImageFFTW[(int)((kx)+size1*(ky))].x*diffImageFFTW[(int)((kx)+size1*(ky))].x+
   diffImageFFTW[(int)((kx)+size1*(ky))].y*diffImageFFTW[(int)((kx)+size1*(ky))].y );
```

4.3.2 Perspective Imaging

The perspective imaging algorithm iterates through either the radiance or the raw plenoptic image. As seen in the previous perspective imaging algorithms, the plenoptic image rendering method for perspective imaging requires obtaining plaid coordinates through a 2-D interpolation using the coordinates of the microlens centers. A kernel call for 4-D interpolation on those newly calculated plaid pixels then calculates the final perspective image. Some overhead is inherent in this algorithm, since identical calculations can occur when calculating the plaid coordinates. However, a reduced calculation time is still seen due to not needing to calculate the entire radiance image.

The perspective imaging algorithm that uses the radiance calls the 4-D interpolation kernel directly on the radiance. The calculation step for the perspective image rendering is therefore less than that of the plenoptic image perspective algorithm. Unfortunately, the calculation of the radiance is a significant factor in the processing time, as shown in Section 4.3.2.1.

The use of the shifted radiance, which is typically used for the FFT-based refocusing algorithm, presents a different problem. Each index in the 4-D radiance will either be a positive value or a negative value. For whole values of (u, v), extracting the correct perspective image is simply an absolute value of the index in the radiance, as shown in Equation (4.32).

$$I_M(s,t) = |L_F(s,t,u_d,v_d)| \tag{4.32}$$

For non-integer values of (u, v), removing the sign from the shifted radiance must be done before the interpolation step. Equation (4.33) shows linear interpolation for removing the sign on the shifted image.

$$N_{1M}(x, I_1, I_2) = (1 - x) * |I_1| + x * |I_2| \tag{4.33}$$

This 1-D interpolation algorithm is used in 4-D quadrilinear interpolation for removing the negative component. Note that N_{1M} is not equivalent to $|N_1|$ of Equation (3.10).

4.3.2.1 CUDA Perspective Imaging Total Execution Times

Figure 4.9 shows the block diagram and the general case CUDA processing times for rendering the perspective images from the radiance generation. As before, the number in parentheses is the speedup over the serial processing time. The transport time for sending the plenoptic image to the GPU (3 ms) is a large portion of the total processing time, while the transport time to send the much smaller 286×190 image is only 0.05 ms. The time for generating the radiance is 43 times faster on the GPU than the serial time, and the 4-D interpolation step is 19 times faster than the serial time. These large speedup times are a product of the ability of the GPU to quickly calculate interpolations.

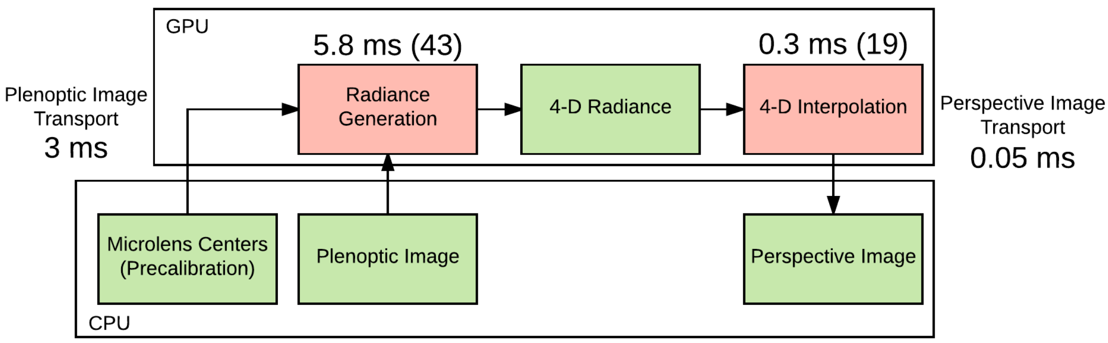


Figure 4.9: Perspective rendering from the radiance with CUDA

Figure 4.10 shows the block diagram for rendering perspective images directly from the plenoptic image using CUDA. In this case, the speedup over the serial case is a significant 43.6. However, the transport time to send the plenoptic image to the GPU dominates the computation time at 5 ms, and therefore the computation time using CUDA is similar to using OpenMP.

Table 4.15 shows the execution times for both the radiance and plenoptic perspective generation methods for the integer and general cases. The plenoptic image transport time for the

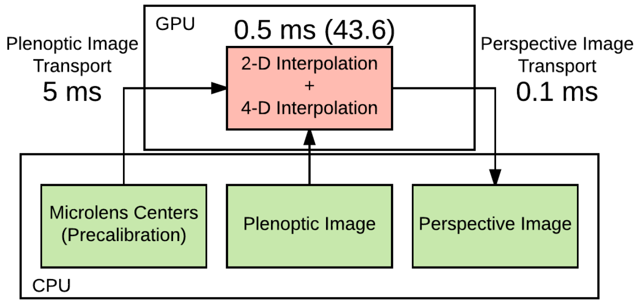


Figure 4.10: Perspective rendering from the plenoptic image with CUDA

plenoptic case is the biggest time commitment, since the perspective algorithm itself is not a significantly huge computational task. The radiance method is more than twice the computational time due to generating the radiance and is only useful if the radiance has already been calculated. The calculation of the radiance is not an advantage over the plenoptic image for successive perspective image calculations (such as rendering other perspectives from a single image) since the plenoptic image transport is not a factor in additional perspective computations.

Table 4.15: CUDA Processing Time for Generating Perspective Image

CUDA Perspective Timing	**(ms)**			
	Radiance		**Plenoptic**	
	Integer	*General*	*Integer*	*General*
Plenoptic Image Transport	3.0004	3.0585	5.0024	5.0024
Radiance Generation	5.8458	5.8458	0	0
Perspective Image Rendering	0.1332	0.2759	0.2056	0.4283
Perspective Image Transport	0.0467	0.0467	0.0660	0.0660
Total	9.0261	9.2269	5.274	5.4967

4.3.3 Integral Refocused Imaging

The integral refocused imaging on the GPU through CUDA is similar to the serial refocused algorithm on the CPU as discussed previously in Section 4.1.3 and will not be repeated here. Its parameters include a pointer to the radiance data structure on the GPU, the output $o \times p$ refocused image, and the α term chosen for refocusing. The radiance data structure has no transport time since the data is already on the GPU. As with most CUDA implementations in this dissertation, most of the `for` loops are replaced by the CUDA indexing within the device code, and the main exception is the summation needed for the integral. Any iterative process such as a summation is not ideal for CUDA, since each of CUDA's threads operate independently from one another in parallel. A shared variable, which is within the scope of every thread in a block, is used for the sum for each of the $o \times p$ grid as shown in Code Listing 4.21.

Listing 4.21: Summation block for each thread

```
__shared__ float sum[16][16];
```

The `__syncthreads()` function waits for each thread to complete before executing the subsequent code. A single thread is used to sum the elements in the $o \times p$ array to get the final summation for the output refocused image. In the code below, the (8,8) thread is tasked with the final summation for the integral in the final refocused image.

Listing 4.22: One thread at (8,8) completes the summation

```
__syncthreads();
if(threadIdx.x == 8 && threadIdx.y == 8)
{ ...
```

4.3.3.0.1 Region of Interest Refocusing Algorithm

One way to reduce the computation time of the integral-based refocusing method is to reduce the size of the rendered image. For some applications such as feature point processing, the entire refocused image does not need to be rendered. By reducing the region of interest for refocusing, the computation time can drop significantly. Similar to Section 4.2.3, the region of interest refocusing algorithm only processes a subset of the image. The `grid` size in this case is reduced to the desired image size. The block indices are also incremented by the x and y offsets to ensure the correct image region is processed. Finally, those offsets are removed when moving the rendered refocused image into the smaller $s \times t$ region.

The region of interest refocusing algorithm takes advantage of the scaling ability of the refocusing algorithm to reduce computation time of small subsets of the refocused image. Table

Table 4.16: CUDA Processing Time for Generating Refocused Image - Region of Interest

Timing	**(ms)**									
Scale	*.05*	*.10*	*1.5*	*.20*	*.25*	*.30*	*.35*	*.4*	*.45*	*.50*
Parallel	0.9	1.9	2.8	3.8	4.7	5.7	6.6	7.6	8.5	9.5
Parallel (Integer)	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
Scale	*.55*	*.60*	*.65*	*.70*	*.75*	*.80*	*.85*	*.90*	*.95*	*1.0*
Parallel	10.4	11.4	12.3	13.3	14.3	15.3	16.2	17.2	18.1	18.9
Parallel (Integer)	4.4	4.8	5.2	5.6	6.0	6.4	6.8	7.3	7.6	8.1

4.16 shows the parallel computation time using CUDA for calculating a small subset of the image. For this data, the *Scale* term refers to the $s \times tn$ size of the image subset, where n is the scale factor used in Table 4.16. For each additional tn, or 2717 pixels, of imagery, the execution time for the general case is about 1 microsecond. For the integer case, the execution time is about half a microsecond. Serial times can be found in Table 4.8.

4.3.3.1 CUDA Integral Refocused Imaging Total Execution Times

Figure 4.11 shows the steps for the integral refocusing algorithm using CUDA. The speedup for the radiance generation is almost 90, while the actual integral refocusing algorithm has a speedup of almost 31 over the serial case. Note that the microlens centers have been previously found and are sent to the GPU once at startup.

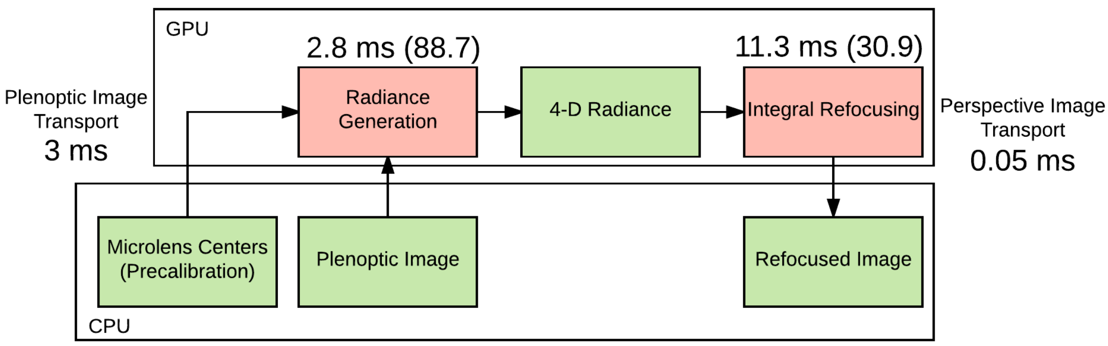


Figure 4.11: Integral refocusing with CUDA

Table 4.17 shows the execution times for each of the integral refocused imaging types. The image transport times are about 4 milliseconds as seen in the other algorithms. The radiance generation, which consists primarily of 2-D interpolation, makes up almost 6 milliseconds. One important point is that the radiance created here is of datatype `float` rather than the complex `float` datatype required for the FFT-based refocusing method. The bulk of the calculation time consists of the refocused image rendering that iterates through the radiance data structure and calling 4-D interpolation kernels on the data. For integer α terms, the 4-D interpolation kernels are not needed, and the computation gain can be seen in Table 4.17. Finally, the transport time needed for moving the rendered refocused image back to the CPU is less than 1 millisecond, which shows the advantage of sending significantly fewer bytes. A single refocused image takes slightly more than 17 milliseconds to process using CUDA. While this is significantly better than the multi-threaded case, any plenoptic-based algorithm for generating point clouds will need to generate multiple refocused images for a single plenoptic image. The FFT-based refocused imaging algorithm addresses this problem.

Table 4.17: CUDA Processing Time for Generating Integral Refocused Image

Refocused Timing	**(ms)**	
	General	*Integer*
Plenoptic Image Transport	3.0578	3.0585
Radiance Generation	2.8168	2.8152
Refocused Image Rendering	11.3391	6.6807
Refocused Image Transport	0.0486	0.0503
Total	17.2623	12.6047

4.3.4 FFT Refocused Imaging

The FFT refocused method for CUDA uses the Fourier slice method to generate refocused images. In addition to the steps described in Section 4.2.4, two additional steps are required to facilitate data transport between the CPU and the GPU. The FFT-based refocusing method is as follows:

- Plenoptic Image Transport
- Shifted Radiance Generation
- 4-D FFT of Shifted Radiance

- 2-D Slice
- 2-D iFFT of Slice
- Conversion of complex data type to `float`
- Refocused Image Transport

The plenoptic image transport to and from the GPU was described in Section 4.3.1.2. The following sections will describe the additional steps.

4.3.4.1 Shifted Radiance Generation

The radiance generation for the CUDA-based refocusing follows the serial and multi-threaded algorithms by creating a shifted radiance to eliminate an additional iteration through the radiance in the Fourier domain. The shifted radiance generation uses the same Fourier property to easily calculate the shifted radiance. The shifted radiance must be used by cuFFT in order to calculate the 4-D Fourier transform of the radiance, and so the shifted radiance data structure is of data type `cufftComplex`, which is a 64-bit data structure made up of two 32-bit `float` data types used to represent the complex data needed for the Fourier transform. This large data structure is represented only on the GPU and so no transport costs are needed to move the shifted radiance to the CPU and is elaborated further in the next section. The execution time for the shifted radiance execution is slightly more than the non-shifted version consisting of 6 ms.

4.3.4.2 4-D FFT Implementation on CUDA

Data transfer from the GPU to the CPU requires non-negligible time. Any implementation of the FFT-based refocused rendering requires an implementation of the FFT on the GPU to avoid unnecessary data transfers. Fortunately, Nvidia offers cuFFT [67], an implementation of the FFT based on FFTW. Unlike FFTW, cuFFT only provides a 1-D, 2-D, and 3-D FFT. To implement a 4-D FFT on the GPU, two batch 2-D FFT's are called, one which implements a (u$\times$v) transform and one that implements a (s$\times$t) transform with a indexing permutation between the two transforms. A final indexing permutation completes the 4-D FFT implementation.

This section describes the four steps of the implementation of this 4-D FFT with CUDA on the GPU. The radiance is contained within a 1-D array of `fftw_complex`, which is bit compatible with CUDA's `cufftComplex` and the standard C++ library's `std::complex<float>`. The `cufftComplex` data structure is used to send the data to the GPU. Those steps are as follows:

- Batch 2-D FFT over $u \times v$
- Permutation from $s \times t \times u \times v$ to $u \times v \times s \times t$
- Batch 2-D FFT over $s \times t$
- Permutation from $u \times v \times s \times t$ to $s \times t \times u \times v$

The first step of the 4-D FFT is a batch 2-D FFT over each of the microlens images, which results in a total of mn or 54,340 2-D FFT operations on size $o \times p$ or 16×16 images. The 2-D representation after this first step on the 4-D radiance can be seen in Figure 4.12.

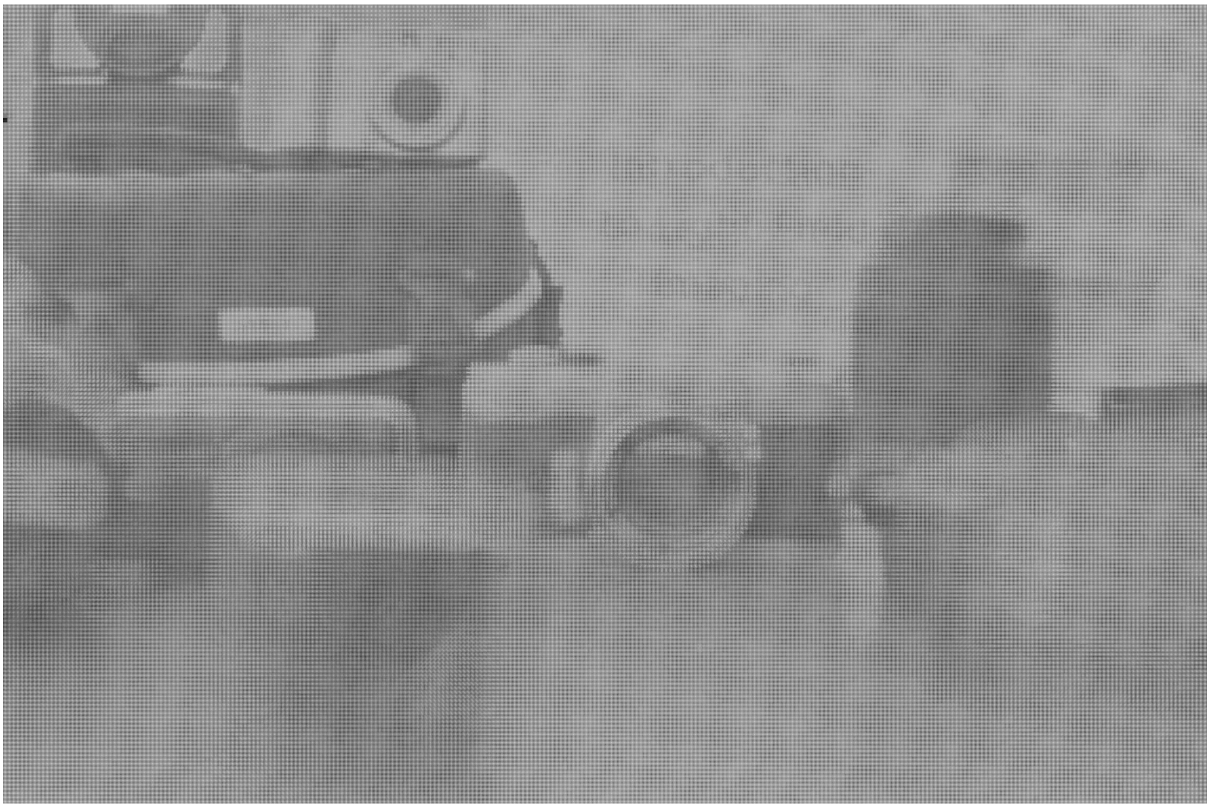


Figure 4.12: 2-D (u,v) FFT (s,t,u,v)

The batch FFT operation requires no device code and is executed using the `cufftExecZ2Z` command which uses the batch fft plan `cufftPlanMany`. The `cufftPlanMany` is constructed according to the Nvidia API as given in Code Listing 4.23:

Listing 4.23: Batch FFT Execution

```
cufftPlanMany(&plan, RANK, NX, &iembed, istride, idist,&oembed, ostride, odist, plan_type,
    BATCH);
```

where for this transform:

$$
\begin{aligned}
RANK &= 2\\
NX &= 16{,}16\\
iembed &= \text{NULL}\\
istride &= 1\\
idist &= 256\ (16\times 16)\\
oembed &= \text{NULL}\\
ostride &= 1\\
odist &= 256\ (16\times 16)\\
plan\_type &= \text{CUFFT\_Z2Z}\\
BATCH &= 54{,}340\ (190\times 286)
\end{aligned}
$$

Figure 4.13 shows the second step in the 4-D implementation on CUDA: the permutation from (s,t,u,v) to (u,v,s,t). This permutation is effectively a non-interpolated perspective imaging over the entire $o \times p$ microlens space, which results in a $o \times p$ number of images, as seen in Figure 4.13. The host call for the transpose is seen in Code listing 4.24.

Listing 4.24: Permutation host call

```
dim3 grid(286, 190, 1);
dim3 threads(16, 16, 1);
fftCudaTranspose<<<grid,threads>>>(radiance,dev_radiancePermute);
```

This actual permutation is done on the GPU in device code. The device code is shown in Code Listing 4.25.

Listing 4.25: Permutation for 4-D FFT to (u,v,s,t)

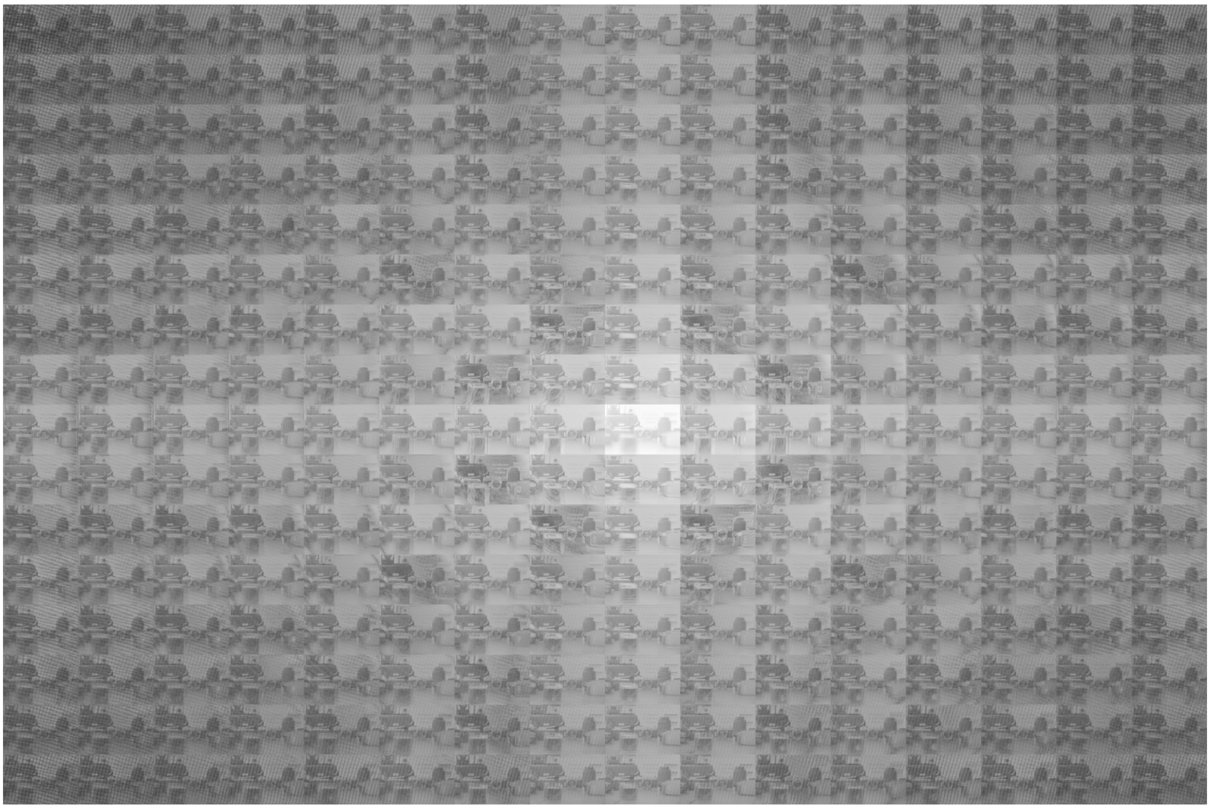


Figure 4.13: (s,t,u,v)$\Longrightarrow$(u,v,s,t) permutation

```
int s = threadIdx.x;
int t = threadIdx.y;
int u = blockIdx.x;
int v = blockIdx.y;

int size1 = gridDim.x;
int size2 = gridDim.y;
int size3 = blockDim.x;
int size4 = blockDim.y;

unsigned int xIndexOp = v+size2*(u+size1*(t+size4*s));
unsigned int xIndex = t+size4*(s+size3*(v+size2*u));
dev_radiancePermute[xIndexOp].x = (float)dev_radiance[xIndex].x;
dev_radiancePermute[xIndexOp].y = (float)dev_radiance[xIndex].y;
```

Figure 4.14 shows the 2-D FFT of the 4-D implementation over dimensions (s,t). This batch 2-D FFT, which results in a total of mn or 54,340 2-D FFT operations on size $o \times p$ or 16$\times$16 images. The 2-D representation after this first step on the 4-D radiance can be seen in

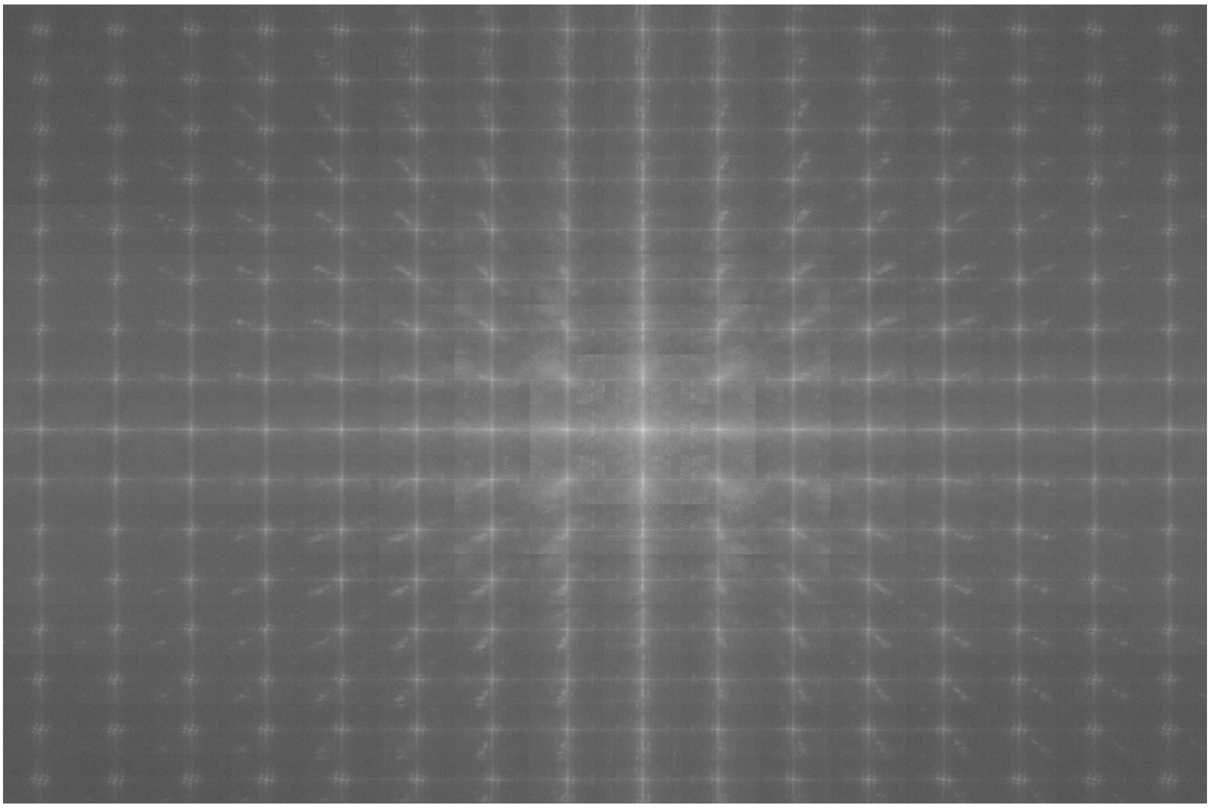


Figure 4.14: 2-D (s,t) FFT (u,v,s,t) permutation

Figure 4.12. This batch FFT operation is also called using *cufftExecC2C* command using the batch FFT plan *cufftPlanMany*. For this batch FFT:

$RANK$ = 2

NX = 286,190

$iembed$ = NULL

$istride$ = 1

$idist$ = 54,340 (190× 286)

$oembed$ = NULL

$ostride$ = 1

$odist$ = 54,340 (190× 286)

$plan\_type$ = CUFFT_C2C

$BATCH$ = 256 (16× 16)

Both 2-D batch FFT operations are more efficient than a series of FFT's due to the reuse of twiddle factors, which are the trigonometric constant coefficients in the FFT algorithm, for each execution of the FFT [43].

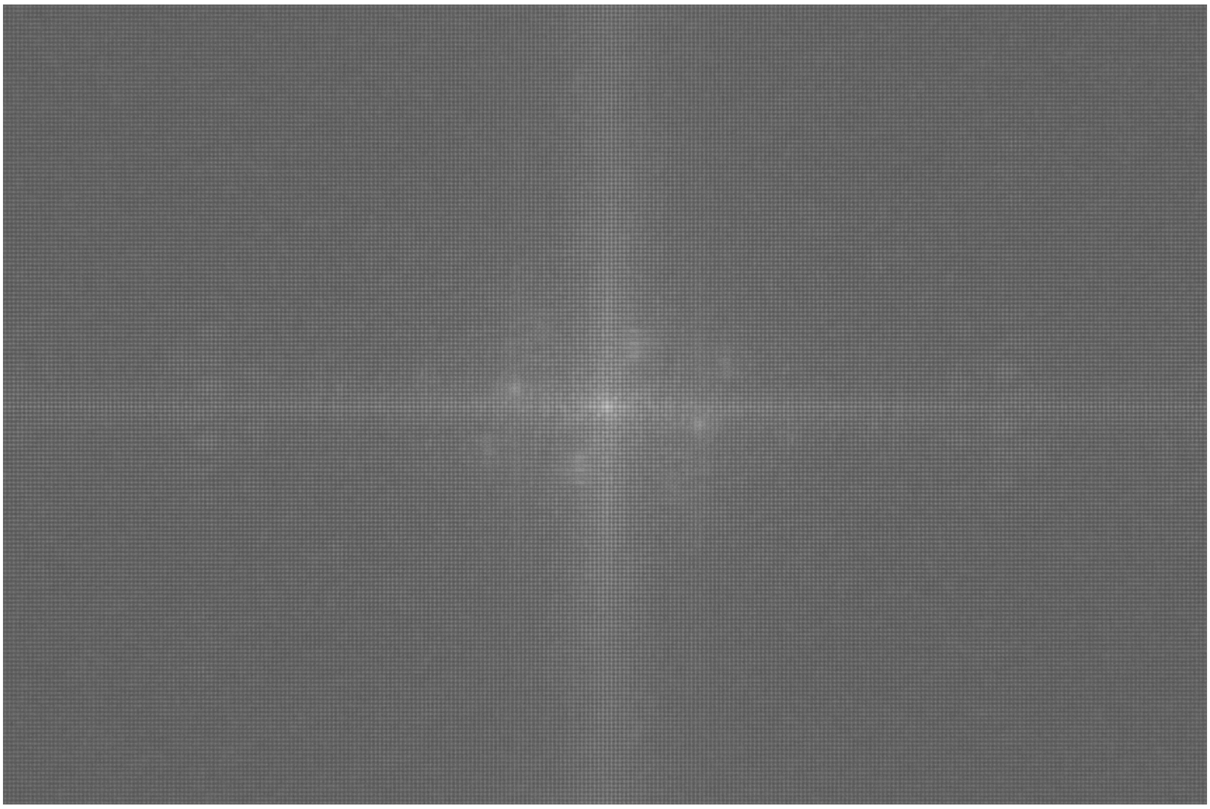


Figure 4.15: (u,v,s,t)$\Longrightarrow$(s,t,u,v) permutation: 4-D FFT of radiance

Figure 4.15 shows the permutation back from (u,v,s,t) to (s,t,u,v), and Code Listing 4.26. This permutation is used to "undo" the first permutation to ensure the correct ordering of the indexing variables. It reorders the 1-D data structure holding the 4-D radiance so that the least significant indexing variables are of size $o \times p$ like the original radiance. The image of Figure 4.15 is effectively a $m \times n$, or 286x190, array of $o \times p$, or 16x16, images in the Fourier domain.

Listing 4.26: Permutation for 4-D FFT back to (s,t,u,v)

```
int s = blockIdx.x;
int t = blockIdx.y;
int u = threadIdx.x;
int v = threadIdx.y;

int size1 = gridDim.x;
int size2 = gridDim.y;
int size3 = blockDim.x;
```

```
int size4 = blockDim.y;

unsigned int xIndexOp = v+size4*(u+size3*(t+size2*(s+size1*0)));
unsigned int xIndex = t+size2*(s+size1*(v+size4*(u+size3*0)));

dev_radiancePermute[xIndexOp].x = (float)dev_radiance[xIndex].x;
dev_radiancePermute[xIndexOp].y = (float)dev_radiance[xIndex].y;
```

Note that the shifting operation was already compensated for in the original generation of the radiance during the interpolation step, thus no -1 multiplier is needed. Originally, this method was chosen to avoid an additional iteration through the 4-D matrix when calling the 4-D FFT through FFTW, as that method did not provide an interface inside FFTW to manipulate the 4-D FFT. However, the shifting operation here can be done in one of the permutation operations if necessary. The execution time for the 4-D FFT operation is by far the most computationally expensive operation in the FFT-based refocused image pipeline for CUDA and making up about 40 ms of the total execution time.

4.3.4.3 Fourier Slice

The Fourier Slice portion of the FFT-based algorithm is similar to the algorithm described in Section 4.1.4. The Fourier slice kernel calculates the pixel values for the sliced image through an $m \times n$ iteration and is indexed as described previously in Section 4.3. To explore the effect of other thread block sizes, a series of different thread block sizes were run, and the execution times are shown in Table 4.18 for the Fourier slice as well as the conversion to `float` type. Note that the maximum block size here is made up of 256 threads, which is a quarter of the maximum threads size of 1024. The block sizes greater than 256 all reported larger execution times. The minimum time consists of a thread block size of 9, or 3$\times$3. Table 4.19 shows the execution time for the Fourier slice kernel for a thread block size of 8.

4.3.4.4 2-D Inverse FFT Implementation on CUDA

The 2-D inverse FFT implementation uses cuFFT. For the cuFFT plan, the width and height are specified along with the desired datatype flag. Since the base datatype is a `cufftComplex`,

Table 4.18: Processing Time for Slicing 4-D - Variable Thread Block Sizes

Slice Timing	**(ms)**							
Thread Block Size	*1×1*	*2×2*	*3×3*	*4×4*	*5×5*	*6×6*	*7×7*	*8×8*
Fourier Slice	0.3555	0.3071	0.2145	0.2444	0.2257	0.2444	0.2703	0.2562
Float Conversion	0.1366	0.1367	0.1360	0.1359	0.1360	0.1373	0.1372	0.1374
Thread Block Size	*9×9*	*10×10*	*11×11*	*12×12*	*13×13*	*14×14*	*15×15*	*16×16*
Fourier Slice	0.2897	0.3147	0.3218	0.3214	0.3576	0.4306	0.4347	0.4560
Float Conversion	0.1359	0.1359	0.1359	0.1363	0.1377	0.1502	0.1685	0.1681

Table 4.19: CUDA Processing Time for Slicing 4-D - Block Size 8

Fourier Slice Timing	**(ms)**	
	General	*Integer*
Fourier Slice	0.2145	0.1816

the desired datatype flag is `CUFFT_C2C`. Code Listing 4.27 shows the cuFFT plan call. Note that this operation is not a batch operation as used in the 4-D FFT CUDA algorithm.

Listing 4.27: Datatype Conversion

```
cufftPlan2d(&ifftPlan, 190, 286, CUFFT_C2C);
```

The FFT call is shown in Code Listing 4.28. Note the important flag in this call to specify the inverse, `CUFFT_INVERSE`.

Listing 4.28: Datatype Conversion

```
cufftExecC2C(ifftPlan, dev_ifftImage, dev_image2, CUFFT_INVERSE);
```

The execution time for the 2-D inverse FFT is about 0.08 ms.

4.3.4.5 Zero Padding

The CUDA execution time for the 4-D FFT is a significant portion of the FFT-refocusing method. One way to speed up the 4-D FFT is to zero-pad the (s, t) time series. For example, the radix-2 Cooley-Tukey FFT algorithm can greatly increase the speed of the FFT for powers of two [41]. Since the (u, v) 2-D FFT is already a power of two, its computation is much faster than the (s, t) computation, as seen in Table 4.20. Unfortunately, the size required for a power of 2 zero padding for the (s, t) FFT is 512×256, which is significantly larger than the non-zero case of 286×190. For the 4-D operation, the transpose for the 512×256 case eliminates any benefit from using a power of two for zero-padding size, as shown in Table 4.20.

Table 4.20: Execution Time for FFT-based Refocusing for Sample Zero-padded Sizes

Zero-padded Timing	**(ms)**			
	(286,190)	(288,192)	(328,226)	(512,256)
Plenoptic Image Transport	4.3	4.3	4.3	4.3
Shifted Radiance	2.9	3.6	4.2	5.1
4-D FFT				
UV 2-D FFT	3.4	3.4	4.4	8.0
Transpose (s,t,u,v)→(u,v,s,t)	13.5	14.0	17.7	31.1
ST 2-D FFT	12.5	3.6	64.1	11.9
Transpose (u,v,s,t)→(s,t,u,v)	12.0	12.2	16.1	27.9
Fourier Slice	0.2	0.5	0.6	1.1
2-D inverse FFT	0.1	0.03	0.3	0.05
Rendered Image Transport	0.05	0.05	0.05	0.05
Total	48.9	41.7	111.8	89.4

4.3.4.5.1 Other Zero-Padded Sizes

The computational benefit of zero-padding is more difficult for other sizes. Since the `cuFFT` algorithm determines the best FFT method for a given problem through the use of a `fft_plan`, the execution times for calculating the FFT-based refocused image can vary wildly. Figure 4.16 shows a color map matrix of the computation times (averaged over a 1000 runs) for the zero-padded FFT refocused method where dark blue is small execution times, and bright yellow are higher execution times. Each axis is an additional two indices for the zero-padding. The index of $(1, 1)$, or the upper left corner, is the case with no zero-padding of size $(286, 190)$, while the lower right corner is zero-padded by 42 indices (each block increments by 2) in each direction with a size of $(328, 232)$. The impact of various zero-padded lengths on the speed of the FFT is seen in the gridded appearance of the plot. The gradual increase in computation time due to the increased time from the transpose can also be seen from the increase in computation time at higher zero-padded sizes.

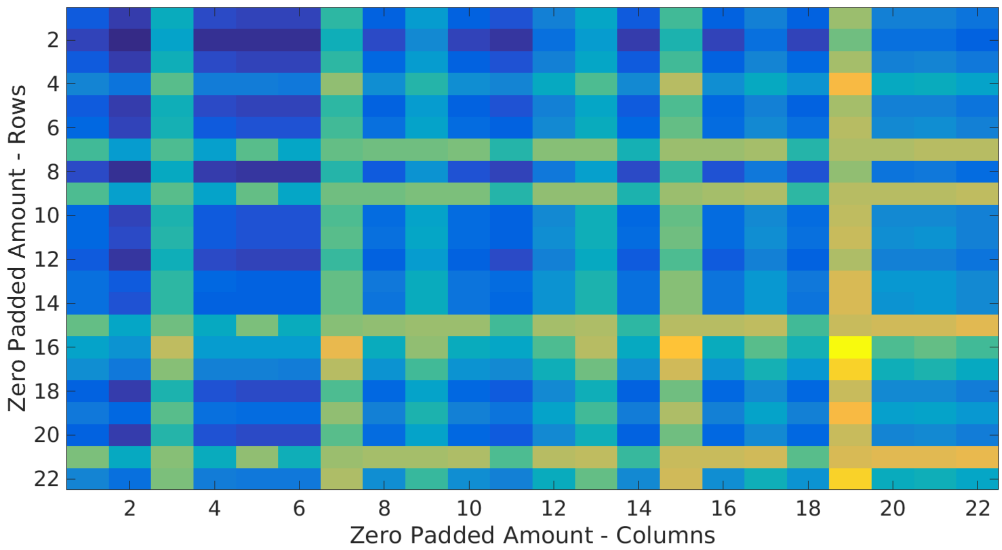


Figure 4.16: Color map matrix of execution times

The smallest computation time (dark blue in Figure 4.16) occurs for (s, t) lengths of 288×192 with a time of 41.9043ms. The zero-padding consists of only two indices in each dimension. While these lengths are not a power of two, both 288 and 192 are regular numbers with factors of 2 and 3 and are easier to calculate for the FFT. The largest computation time (bright yellow) in the execution time matrix is at size $(316, 226)$, which corresponds to an execution time of 111.9 ms. 316 has prime factors of 2 and 79, while 226 has prime factors of 2 and 113, which shows the difficulty in execution for the FFT to easily divide the transform into smaller FFTs, as seen in Table 4.20 with a 2-D (s, t) FFT computation time more than five times that of the zero-padded case. The execution times of Figure 4.16, rounded to the nearest millisecond, can be found in Appendix B.

4.3.5 CUDA FFT-based Refocused Imaging Total Execution Times

Figure 4.17 shows a block diagram of the FFT-based refocusing method as implemented on the GPU using CUDA. The shifted radiance generation has a speedup of 88, which is similar to the integral method radiance generation. For the 4-D FFT, most of the computation time is spent in the permutation step. The 2-D FFT for the (u, v) coordinates is 3.4 ms, while the computation time for the (s, t) coordinates is significantly longer at 12.5 ms. This disparity is due to the 16×16 block for the (u, v) case, which is much faster for the FFT as a power of

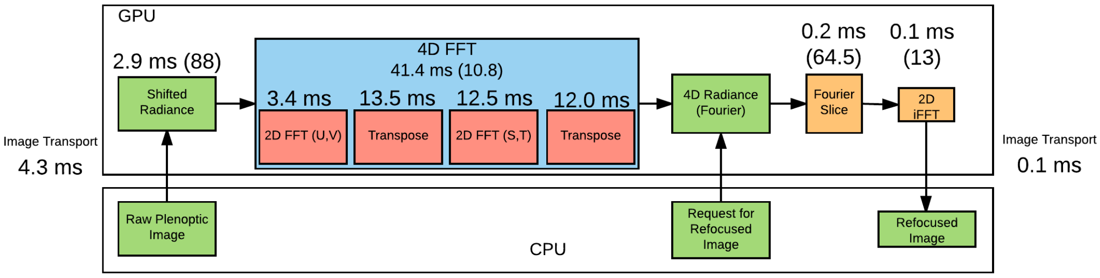


Figure 4.17: FFT Refocusing with CUDA

2. The Fourier slice and 2-D inverse FFT are both very fast on the GPU with speedups of 64.5 and 13, respectively.

Table 4.21 shows the processing times for the steps involved in computing the FFT-based refocused image for the no zero-padded case. The 4-D FFT is the largest computational cost in the refocused imaging pipeline. The rendered image transport includes the time required to convert the `cufftComplex` type to `float`.

Table 4.21: Processing Time for FFT-based Rofocusing

Refocused FFT Timing	**(ms)**	
	General	*Integer*
Plenoptic Image Transport	4.35	4.42
Shifted Radiance	2.88	2.88
4-D FFT		
UV 2-D FFT	3.36	3.36
Transpose (s,t,u,v)→(u,v,s,t)	13.49	13.49
ST 2-D FFT	12.51	12.51
Transpose (u,v,s,t)→(s,t,u,v)	11.99	11.99
Fourier Slice	0.21	0.18
2-D inverse FFT	0.08	0.09
Rendered Image Transport	0.05	0.05
Total	48.92	48.97

4.4 Rendering Imaging Results

The overall best times for the perspective, integral refocused, and FFT refocused image rendered algorithms for the serial, OpenMP, and CUDA architectures are shown in Table 4.22.

Table 4.22: Processing Times

Execution Times	**(ms)**	
	Integer	*General*
Perspective Imaging		
Serial	3.5	21.8
OpenMP	1.3	5.6
CUDA	5.3	5.5
Integral Refocusing		
Serial	196.5	699.1
OpenMP	111.0	242.2
CUDA	12.6	17.3
FFT Refocusing		
Serial	706.4	717.3
OpenMP	212.7	217.4
CUDA	49.0	48.9

4.4.1 Perspective Imaging

The performance of the perspective imaging algorithms was heavily dependent on whether or not an integer case, where (u, v) were both integers, was rendered. Due to the lack of computation needed for perspective rendering, the CUDA case resulted in the worst perspective times for the integer case because of the transport time from the CPU to the GPU, which made up the vast majority of the calculation time. The fastest integer case calculation came from the OpenMP case, which used its multithreaded capabilities to drop its execution times below the execution time of the serial case.

In contrast, the CUDA implementation for the general case, where either u or v are not integers, is the algorithm with the fastest times. The serial and OpenMP execution times could not drop below the transport times of the CUDA implementation. The serial algorithm also reduced its own computation time to well above the OpenMP and CUDA implementations, which shows the benefit of parallel processing for interpolation problems.

4.4.2 Refocused Imaging

The integral and FFT-based refocused imaging algorithms are effectively interchangeable. Their algorithms compete for the same role to render a refocused image. Table 4.22 shows that the FFT-based refocusing has very little difference between the integer and general case, since the α term is only used in the Fourier slice algorithm, which is computationally light. The integral refocusing, however, has a significant difference between the integer and general case for the serial, OpenMP, and CUDA algorithms. The most efficient algorithm for the integral refocusing case is the CUDA case due to its massively parallel capabilities. The CUDA case is also most efficient for the FFT refocused imaging algorithm and improves on the OpenMP time execution by almost a factor of five. The slowest case for refocusing is the general, i.e. non-integer, serial case, as expected with almost an entire processing time of one second. Note that this time is actually faster than the single core OpenMP execution time since no OpenMP overhead was present to slow the processing time of the serial case.

The fastest image rendering algorithm is the CUDA integral refocusing algorithm, which improves upon the FFT-based refocusing algorithm by almost twice the speed. However, each successive refocusing requires another full execution (about 50 ms) to render another refocused image. For plenoptic point cloud generation, a refocused stack of images are needed. The FFT refocused imaging algorithm, while slower than the integral refocusing method due to its significant 4-D FFT processing time, can very quickly slice additional refocused images to create a focal stack at a third of a millisecond. The CUDA-based FFT refocusing algorithm, then, is the most efficient algorithm to choose for creating more than two refocused images from a single plenoptic image.

4.4.3 Future Improvements

Additional modifications are possible that could improve the performance and quality of the rendered images. The interpolation method used in all of these time execution times was either quadrilinear or bilinear interpolation, depending on the dimension. A better quality interpolation method can greatly improve the quality of the resulting images, including reduced aliasing, at the cost of greater execution time. Several image feature enhancements can be also added to the image rendering algorithms. Additionally, the use of the inverse interpolation curve can reduce vignetting, and duplicating the image beyond its bounds can reduce edge aliasing.

Chapter 5

Plenoptic Visual Odometry with Sparse Point Clouds

Plenoptic image processing in real-time as described in Chapter 4 is one important part of using plenoptic cameras in vision navigation. Another important part is how these rendered images can be used to generate sparse point clouds. Point clouds are a commonly researched topic in a variety of navigation algorithms [68]. In this dissertation, the vision navigation algorithm chosen for implementation of these plenoptic point clouds is visual odometry. Visual odometry has a wide range of uses and implementations such as Simulataneous Localization and Mapping (SLAM) and structure from motion (SfM) [5]. The goal of visual odometry as defined in this dissertation is ultimately the ability to track the rotation and translation of a camera from one image to another image using the point clouds generated from the plenoptic camera. This chapter explores current visual odometry algorithms using point clouds in monocular and stereo motion. It also describes proposed algorithms for plenoptic point cloud generation using the unique rendering capabilities of the plenoptic camera.

5.1 Monocular Visual Odometry

The cost of multiple cameras over a single camera is a significant concern for some applications. However, a single traditional camera cannot provide depth information from one image without some knowledge of the environment. Monocular cameras can provide what is known as bearing-only Simultaneous Localization and Mapping (SLAM), which provides course [5]. Another method of using monocular cameras uses the parallax from a moving camera to determine the depth to a camera either over time or through a batch process using recorded feature points. In these situations, the multiple images over time from the single camera replace the multiple cameras from a single time in a stereo or multi-camera setup. This approach is typically combined with a SLAM approach to track features in 3-D and reduce the uncertainty of its position over time [69, 68]. Semi-dense methods have also been proposed that estimate a depth map of the entire image [70]. These monocular camera systems perform best in a static environment to maintain the estimation of depth to the tracked features over a series of frames. In contrast, stereo odometry approaches can immediately obtain depth through triangulation, as discussed in the next section.

5.2 Stereo Visual Odometry

Stereo visual odometry is a well researched area in robotics and vehicular navigation. A full survey of the many methods proposed to solve the visual odometry problem is outside the scope of this dissertation. This section, instead, will focus on widely adapted methods for visual odometry that are mature enough to warrant inclusion in open source software. At its core, stereo odometry uses triangulation between two cameras to obtain depth. Usually, the two cameras are separate cameras that use an electrical trigger to ensure that the images from each of the cameras are taken at the same time. Without the time synchronization of the cameras, the feature correspondence between images will be wrong, which results in an erroneous depth map. Another assumption regarding stereo cameras is that each camera maintains a rigid and constant rotation and translation from each other. These extrinsic properties must not change to ensure proper usage of the stereo calibration that is assumed to be known in this dissertation. In practice, any jolt or mishandling of the stereo camera rig may result in breaking this relative position assumption, which would then require recalibration of the stereo camera.

This next section describes the stereo front end, which consists of the image processing steps required to detect features for the calculation of a depth map used in this dissertation. Section 5.2.2 then describes the stereo back end, which consists of the calculation of the depth to features for a point cloud and the rotation and translation from a point cloud at one point in time to a point cloud at another point in time. Next, a simulation of the stereo back end under perfect conditions is presented.

5.2.1 Stereo Odometry Pipeline Front End

Stereo odometry requires several steps to extract motion information from images through image processing. This dissertation covers feature-based visual odometry, which finds strong edges in images to track from frame to frame. This process reduces computation time by limiting the number of pixels that need to be processed. The first step in the stereo odometry pipeline is compensating for any radial or tangential distortion in the images through the intrinsic and extrinsic calibration parameters, as described previously in Section 2.3.

5.2.1.1 Stereo Rectification

Typically, stereo cameras are aligned in parallel with a translation known as the baseline between them. However, these two cameras may not necessarily be aligned perfectly due to both rotation and translation. A vertical discrepancy can cause misalignment in matches, while rotational differences can alter the perspective to features, as shown in Figure 5.1. In addition, distortion in cameras will induce error in the pinhole model when projecting or reprojecting points to 3-D. These problems can result in erroneous feature matches and poor disparity maps. Stereo rectification, then, aligns the stereo images through warping the images through the known rotation and translation between the cameras as well as accounting for any distortion in the images. It includes:

- Account for distortion through intrinsic parameters
- Adjust image to align the left and right images through extrinsic parameters

The intrinsic parameters are found through intrinsic calibration, as described previously in Section 2.3.1. The rotation and translation used for warping the image is determined through stereo calibration, which was described in Section 2.3.2.

5.2.1.2 Feature Correspondence

Three steps are necessary for determining corresponded features between multiple frames: feature detection, feature description, and feature matching. Feature detection finds strong

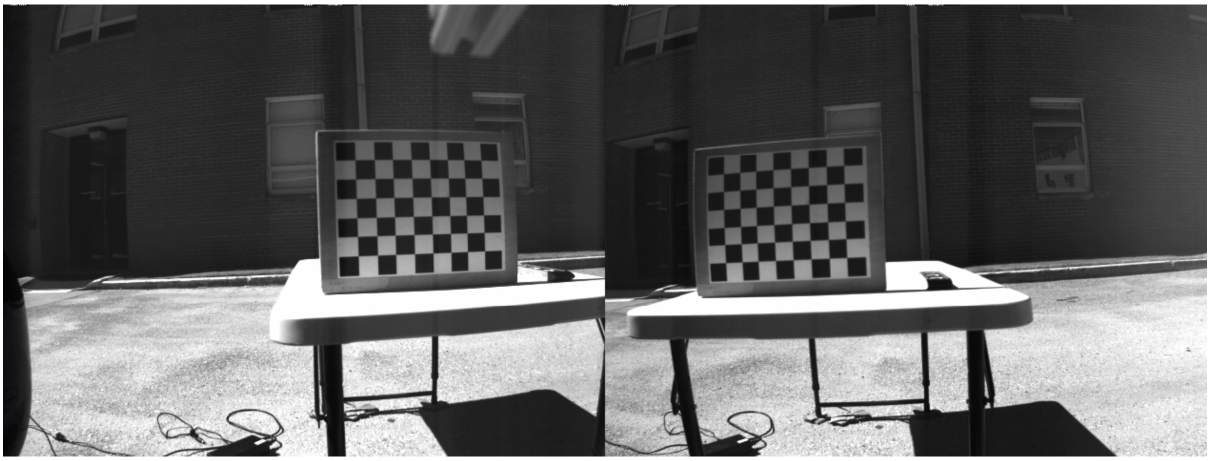


Figure 5.1: A calibration target on distorted and misaligned raw stereo imagery

edges that provide robust information for the feature descriptor. The feature descriptor is the method by which the detected feature is stored and is ideally invariant to any transformation to adequately describe a feature even in large translations or rotations. The feature matcher uses the information from the feature descriptor to match detected features from one frame to another. This dissertation uses feature detectors, feature descriptors, and feature matchers available within OpenCV [71].

5.2.1.2.1 Feature Detection

The feature detection method finds strong corners in an image that will be consistent over multiple images. One popular feature detection method is FAST(features from accelerated segment test) for its computational efficiency and performance. FAST uses a circle of sixteen pixels to classify whether or not a point is a corner [72]. The FAST feature detector has the advantage of speed over other methods as well as its improved performance over more traditional methods such as the detector method by Shi and Tomasi [73].

5.2.1.2.2 Feature Descriptors

Feature descriptors are a widely researched area of study where the strongest edges in images that allows for repeat detection in the presence of rotation and translation are extracted from image to image. Feature descriptors quantify how the feature itself is identified and stored compared to other locations in the image. Binary descriptors are a type of feature descriptor that stores the information of a descriptor patch as a binary string. Their implementation is fast due to their ability to match patches based on the sum of an XOR operation for the hamming distance between two binary strings. BRISK (binary robust invariant scalable keypoints) [74], ORB (oriented FAST and rotated BRIEF) [75], and BRIEF (binary robust independent elementary features) [76] are three common binary descriptors available in the literature. FREAK (fast retina keypoint) uses methods used by the human visual system (the retina) to compute a cascade of binary strings from image intensities over a retinal sampling pattern [77]. Binary descriptors were chosen for their lower memory load and speed, and increased robustness over histograms of gradient-based descriptors such as SIFT (scale-invariant feature transform) [78] and SURF (speeded up robust features) [79]. Many of the binary descriptors have corresponding detectors as well.

5.2.1.2.3 Feature Matching

Feature matching seeks to match the detected features from one image to another image. The matcher method used for this dissertation is a simple brute force method which uses the L1 norm for the descriptor comparison. Figure 5.2 shows two similar images with features extracted and described using ORB. The corresponding features from each image are found using the matching algorithm, and a line is drawn from one feature to its matched pair. Since these two images are horizontally aligned, the lines between matched points are horizontal as seen in Figure 5.2.

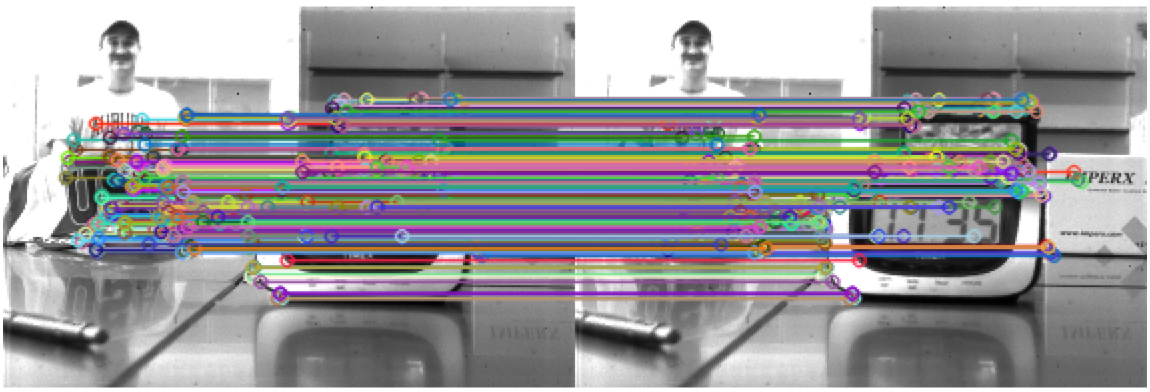
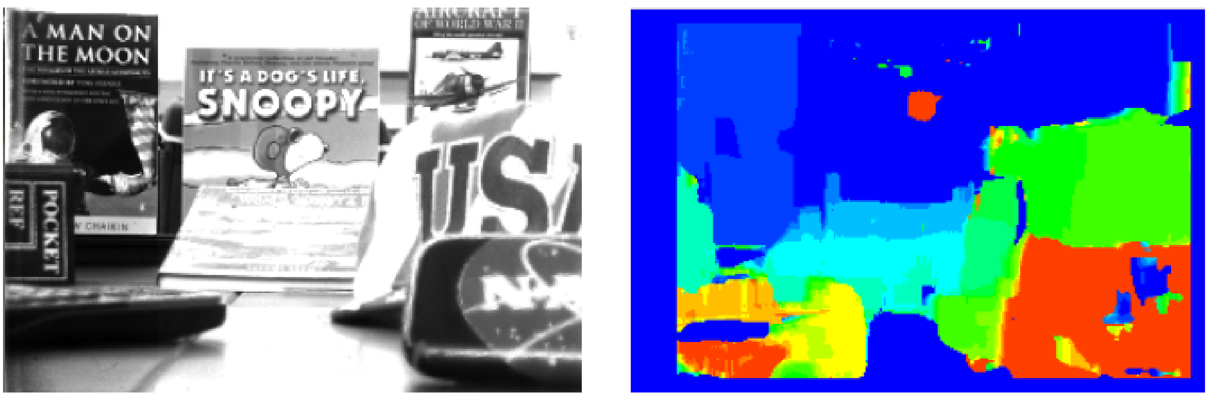


Figure 5.2: Feature matching using ORB detectors, descriptors, and bruteforce L1 matching

5.2.1.3 Disparity Maps

Feature point matching can be augmented with the development of disparity, or depth, maps that consist of the differences between the two point locations in the two camera images. These disparity maps are calculated through stereo correspondence. Some effort has been conducted to evaluate the multitude of stereo correspondence algorithms in literature as seen in [80]. Konolige uses small sum of absolute difference (SAD) windows for relating points between the two stereo images to find the strong matched points [81]. Shiftable-window algorithms such as [82] use a general error filter using the correlation function to invalidate uncertain matches, and a border correction method improves object borders further in post-processing. Graph-cut algorithms have shown promise through computation of local minimum of energy functions using graph cuts [83]. Layered stereo correspondence calculates scene structure using a collection of smooth surface patches, with the disparities modeled as splines [84]. Belief propogation methods use the Bayesian Belief Propagation algorithm to estimate the optimal solution and a probabilistic framework to integrate additional information such as segmentation [85].

A disparity map provides a disparity for most of the pixels in an image. Due to missed overlap or significant rectification, some pixels may not be available in both camera images, and their disparities are unknown. Thus, the cameras in a stereo rig must have significant overlap or the field of view where 3-D point clouds can be created will be very narrow.



(a) Left image - books (b) Book target correlation curves

Figure 5.3: Depth map with plenoptic imagery

The right image in Figure 5.3 shows a colorized depth map of the book image on the left. Darker red colors indicate objects closer to the camera, while bluer objects represent further objects. The objects in this image are all very close, and the depth discrimination capabilities for the near range show the ability of the plenoptic camera to range to various objects through typical stereo camera algorithms. However, the ranges fail to capture the books furthest to the back of the scene. Locations with no features, such as the white reflection in the bottom center of the image, provide no information to the block matching algorithm to calculate a disparity. Additionally, the large red dot in the middle of the blue background shows an interesting failure of the stereo block matching algorithm. The block matching algorithm erroneously matched the "O" text that was next to each other which caused the block matching algorithm to register the disparity at this point as large (nearby). These problems are mainly attributed to issues with the block matching algorithm itself rather than the plenoptic camera. Similar results can be expected from a stereo camera with a small baseline.

5.2.2 Stereo Odometry Pipeline Backend

Once the matched feature points are found, the depth can be estimated using either the disparity map or the difference in pixel coordinates of matched features from the reprojection of points to 3-D. Matching is not limited to the left and right images in a stereo pair. The matching operation can also relate features at one point in time to features at another point in time to extract the change in the position of a feature over time. The motion of a feature over time in an image and its reprojection to 3-D gives the capability of extracting the translation and rotation of the camera. This stereo pipeline backend uses the detected features and their depth from the stereo front-end to estimate the odometry of the camera over time. A simulation of the stereo pipeline backend can be found in Appendix C.

5.2.2.1 Point Cloud Generation

With knowledge of the disparity of a particular feature or point, a point cloud can be generated through standard image projection equations. The depth estimate from disparity comes directly from the geometry of two near-parallel cameras, as shown in Figure 5.4. The

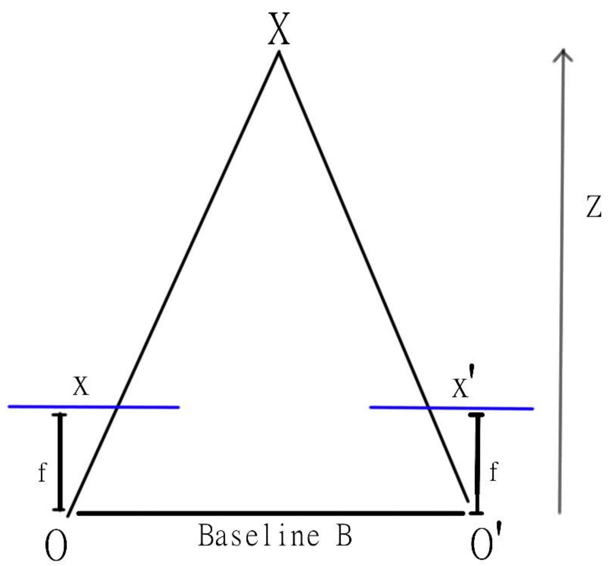


Figure 5.4: Assumptions of stereo camera setup for calculation disparity

blue straight lines in the figure are the image planes, which through rectification are coplanar. With knowledge of the focal lengths and baseline of the stereo cameras, the equation for depth estimation using disparity is seen in Equation (5.1).

$$Z = \frac{Bf}{d} \tag{5.1}$$

where B is the baseline between the cameras, f is the focal length of the cameras, and d is the disparity in pixels. Using Equation (5.1) in homogeneous coordinates where $X_p = \frac{B(x+c_x)}{d}$ and $Y_p = \frac{B(y+c_y)}{d}$ results in Equation (5.2).

$$\begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{B(x+c_x)}{d} \\ \frac{B(y+c_y)}{d} \\ \frac{Bf}{d} \\ 1 \end{bmatrix} = \begin{bmatrix} x+c_x \\ y+c_y \\ f \\ \frac{d}{B} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{1}{B} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = Q_R \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \tag{5.2}$$

The matrix form represents the calculation of the intrinsic parameters, c_x, c_y, and f with the extrinsic translation B. Through rectification, the other intrinsic parameters, both rotation and translation in other axes, are assumed to be negligible.

5.2.2.2 RANSAC

Random Sample Consensus (RANSAC) is a commonly used method for estimating parameters of a model in the presence of noise. In particular, this method separates valid measurements that fit the model from noisy measurements that do not fit the model. Its name comes from the random sample of data used to determine the model parameters, which are compared to a cost function for determining the inliers (consensus set) and outliers of the dataset. If the number of inliers is sufficiently large, the model is considered valid, and the model parameters can be recalculated using the consensus set. A brief overview of the algorithm is listed in Algorithm 1.

Algorithm 1 RANSAC

Require: D contains inliers of some model and outliers due to noise
  for $i = 0 \ldots$ iterations **do**
    $s \leftarrow$ Randomly sample D
    $m = F(s)$ Convert sample s to model with function F
    $p = G(m)$ Determine model parameters from model m with function G
    for $j = 0 \ldots$ size(D) **do**
      $c(j) = C(m(p), D(j))$ Calculate cost c using cost function C
      if $c(j) < t$ **then**
        $I(count) = D(j)$ Add $D(j)$ to list of inliers I
        $count = count + 1$ Increment count of inliers
      end if
    end for
    if $count > count_{last}$ **then**
      $I_{best} = I$ Update current iteration as the best iteration
    end if
  end for

5.2.2.2.1 RANSAC line estimation

As a simple example, RANSAC can be used to separate the inliers and outliers of a datset for estimating a line. In this example, inliers were calculated using the equation of a line plus a small random noise term and outliers were calculated using the same line equation but with a large random noise term. Figure 5.5 shows the set of points used in this example with the true inliers (red) and outliers (blue). Note that the RANSAC algorithm itself has no previous knowledge of which points are inliers or outliers and some of the outlier points have a randomly small noise and can be construed as inliers. For each iteration of the RANSAC algorithm, two points are randomly chosen and used to calculate the slope and y-intercept (the model parameters) for the estimated line (the model). The cost function for determining inliers and outliers is the normal distance from the estimated line. Each point is then chosen as either an inlier or an outlier. If the number of inliers of this model for this particular iteration is larger than the last best model, then the current model parameters are updated as the best model parameters.

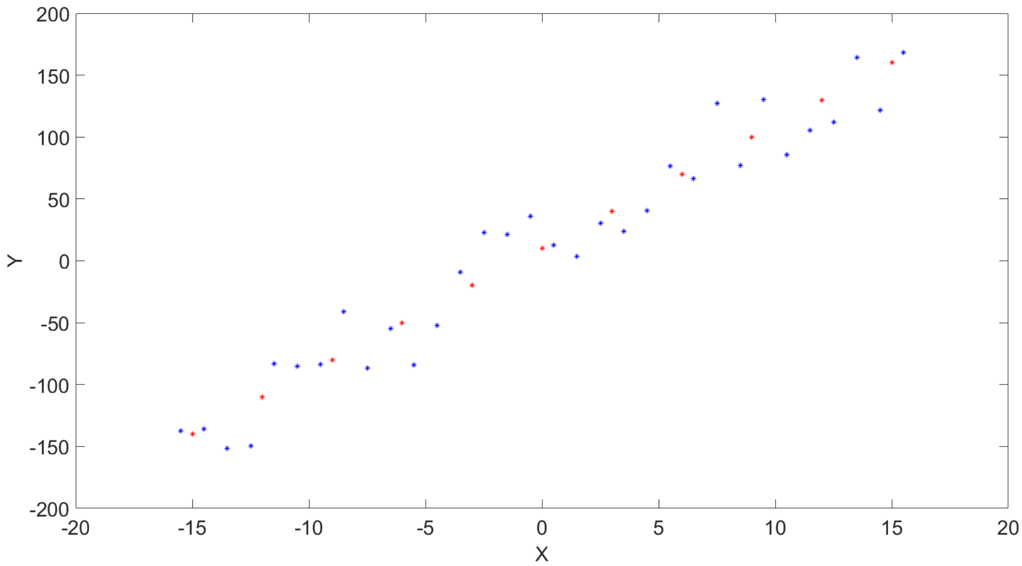


Figure 5.5: RANSAC example: data points

Due to the inherent randomness of the RANSAC algorithm, results can vary over time. In addition, several iterations are required in order to sufficiently sample the dataset to ensure that the best parameters for the model have been found. Figure 5.6 shows the first iteration of the sample model. Since the two points used for estimating the model parameters were chosen randomly, there is no guarantee that the two chosen points were inliers. In this case, one of the points was an inlier, while the other point an outlier. After ten iterations, as shown in Figure 5.7, enough samples have been taken from the dataset that model validity can be reasonably assumed. The cost function, random sample, number of inliers, and necessary iterations required all depend on the problem being solved.

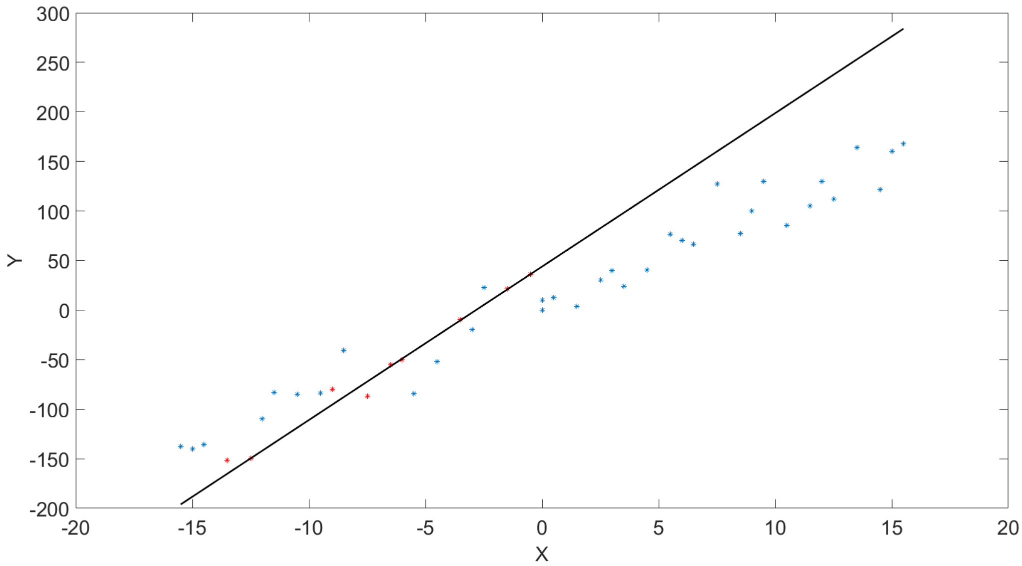


Figure 5.6: RANSAC example: line estimate over one iteration

A few assumptions are made in the RANSAC algorithm. One assumption is that there is only one model present in the dataset. If the random noise on the outliers happens to form another line, RANSAC will fail due to the ambiguity between the RANSAC-calculated inliers from the outlier line and the RANSAC-calculated inliers from the inlier line. Another assumption is that there are a sufficient number of inliers as opposed to outliers for modeling the line. Too many outliers can result in a probabilistically difficult scenario where the inliers are confused with the outliers. Another assumption is that the number of iterations used by the RANSAC algorithm is sufficient. With too few iterations, the randomly chosen data sample may not correspond to the real inliers.

5.2.2.2.2 RANSAC Uses

RANSAC has a wide variety of uses in image processing. RANSAC can be used for reducing erroneous matched feature points and finding the homography of images [86]. It can also be used to find an object pose from 3-D to 2-D point correspondences.

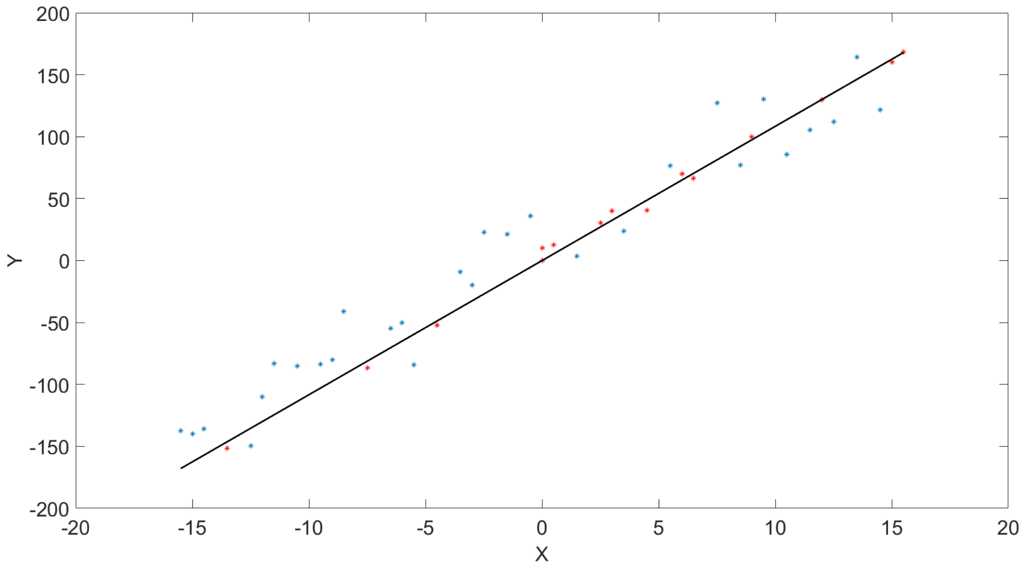


Figure 5.7: RANSAC example: line estimate over ten iterations

5.2.2.3 Pose Estimation

Pose estimation attempts to determine the rotation and translations necessary to match the point cloud from one point in time to the next point in time. This dissertation will describe three of those methods. Ultimately, the accuracy of the point cloud will dictate the accuracy of any of the rotation and translation methods, and any methods that use point clouds regardless of the source (such as lidar or stereo camera) can replace these methods.

5.2.2.3.1 2D to 3D Point Correspondence

The perspective n-point algorithm was developed to compare n image coordinates from one image pair to their corresponding 3-D points in another stereo pair. Gao described the complete solution to the 3 point (P3P) problem for determining the rotation and translation of a camera with respect to a scene object from n corresponded image points [87]. The significance of the three point problem is that three is the smallest subset of correspondences that result in a finite number of solutions. Typically, however, four points are used to eliminate ambiguous solutions. Lepetit extended the solution to an n-point problem through a non-iterative solution with $O(n)$ for more than 3 correspondences (beyond the P3P problem) [88]. Instead of using feature points directly from image processing, Lepetit's work uses the coordinates of the

n points as a weighted sum of four virtual control points. The problem, then, becomes estimating the coordinates of the virtual control points rather than trying to solve for a potentially large n number of feature points. The rotation and translation can then be found using the four virtual control points. Open source software for executing the PnP problem is available, including OpenCV's `solvePnP` and `solvePnPRansac` functions, which were used for this dissertation.

5.2.2.3.2 3D to 3D Point Correspondence

The point clouds from one image pair to another image pair can be used to extract the rotation and translation of a camera. Two methods are discussed here: iterative closest point and singular value decomposition.

Singular Value Decomposition (SVD)

This section describes a process for determining the rotation and translation of corresponded point clouds using the singular value decomposition of a constructed matrix [89]. The SVD approach works well without noise, but with noise, the problem becomes significantly harder [90]. Fundamentally, SVD seeks to minimize the reprojection error from rotating and translating the 3-D points. The first step is determining the centroid of each point cloud. A matrix, H, is created using the sum of the product of the difference between each point in their respective point clouds and the centroid

$$H = \sum_{n=0}^{L} (A_n - \bar{A})(B_n - \bar{B}) \tag{5.3}$$

where $\bar{A}$ and $\bar{B}$ are the centroids of the point clouds A and B, L is the number of corresponded points in the $L \times 3$ point clouds A and B. The singular value decomposition of the H matrix extracts the U,V matrices.

$$[U, S, V] = SVD(H) \tag{5.4}$$

A check for a negative determinant for the matrix V is required to ensure the reflection case is handled.

$$V = \begin{cases} V \text{ for } |V| \geq 0 \\ V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \text{ for } |V| < 0 \end{cases} \tag{5.5}$$

The change in rotation can then be calculated using the U and V matrices.

$$R_C = VU^T \tag{5.6}$$

The translation calculation then uses the centroids and the change in rotation.

$$T_C = R_C\bar{A} - \bar{B} \tag{5.7}$$

Iterative Closest Point (ICP)

Iterative closest point (ICP) is an algorithm that aligns overlapping 3-D point clouds by minimizing the sum of the squared error, shown in Equation (5.8) [89].

$$E(R,t) = \frac{1}{n}\sum_{i=0}^{n} ||x_i - Rp_i - t||^2 \tag{5.8}$$

Unlike the SVD algorithm, ICP does not require correspondence between the point clouds. However, ICP requires overlapping point clouds to ensure convergence to the true minimum. ICP is often used with very dense point clouds such as lidars. The general steps in ICP are as follows [91]:

- Point-per-point matching between point clouds
- Estimate rotation and translation that minimizes Equation (5.8) after rejecting outliers
- Transform using the estimated rotation and translation
- Iterate

Each of these steps have been analyzed in literature to drastically reduce computational time, especially to reduce the time required for the point-per-point matching. Of particular note is Zhang's work, who used a modified K-D tree to reduce the point-per-point computation time [92].

5.2.2.4 Bundle Adjustment

Bundle adjustment [93] is typically the last step in the stereo odometry pipeline. It is basically a large sparse geometric parameter estimation problem, where the parameters consists of the 3-D feature coordinates, the camera poses, and the calibration parameters. Its name originates from the bundle of light rays that converge at a particular point for an image. Bundle adjustment can be used to refine the solution of a visual odometry algorithm or 3-D reconstruction problem. While bundle adjustment can reduce error, it will not be used in this dissertation for the comparison between the perspective and refocused plenoptic odometry methods, but each method can be improved using bundle adjustment.

5.3 Plenoptic Visual Odometry

Light field cameras have not traditionally been used in SLAM because they are thought to increase the amount of data produced and require more computational power [94]. Recent studies have shown that they are particularly suitable for SLAM applications because the motion estimation problem can be reduced to a linear optimization for more accurate motion estimates [94]. One common approach to visual odometry is the generation of a point cloud using any ranging-capable sensor such as a lidar, stereo camera, or, as used in this dissertation, a plenoptic camera. One of the most important tasks for generating a point cloud with any camera is ranging. This section describes point cloud generation with an emphasis on ranging using the plenoptic image rendering techniques presented in Chapter 4.

5.3.1 Depth Discrimination for Visual Odometry

Drazic explored the depth discrimination capabilities of plenoptic cameras with a depth discrimination formula as given in Equation (5.9) [95].

$$\frac{\Delta z}{d_0} \approx \pm \frac{np}{A}\frac{d_0}{f} \tag{5.9}$$

If $f \approx \frac{D}{\alpha}$ and $A = \frac{f}{k} \approx \frac{D}{\alpha k}$, then

$$\frac{\Delta z}{d_0} \approx \pm Lk(\frac{\alpha}{D})^2 d_0 \tag{5.10}$$

where D is the size of a lens from the microlens array, α is the field of view, d_0 is the distance of a point object from the entrance pupil of the main lens, k is the f-number, L is the size of a lens from the array, and Δz is the distance by which a point object is moved from its initial position (at d_0) for generating a disparity of 1 pixel. Generally, a smaller f-number, or k, is better for depth discrimination since small values of k also mean larger lens apertures, which increases the baseline of the camera, as seen in Figure 5.8. In this figure, the relative depth discrimination is a percentage of the initial depth, d_0, that is needed to calculate a disparity of one. Therefore, a smaller relative depth discrimination means that a smaller change in depth is detectable as desired. Drazic also mentions the loss of depth discrimination at larger distances just as in any other depth measuring camera system [95]. Depth discrimination can also be achieved with a small pixel pitch, since $L = np$ where L is the lenslet size and n is the number of pixels per lenslet (also called the elemental image) in one dimension, at the cost of diffraction problems. Unfortunately, a small n results in a low resolution rendered image, which makes choosing L a significant tradeoff between high resolution and depth discrimination. The easiest solution is to have a large sensor area for enough n as well as depth discrimination. Drazic's conclusion was that the two most important parameters of Equation (5.9) are the field angle α and the microlens size D, since they both influence the depth discrimination in a nonlinear manner, as seen in Figure 5.9 [95]. A larger field angle results in a larger depth change to be detectable, while a smaller sensor size provides less capability to detect a change in disparity [95].

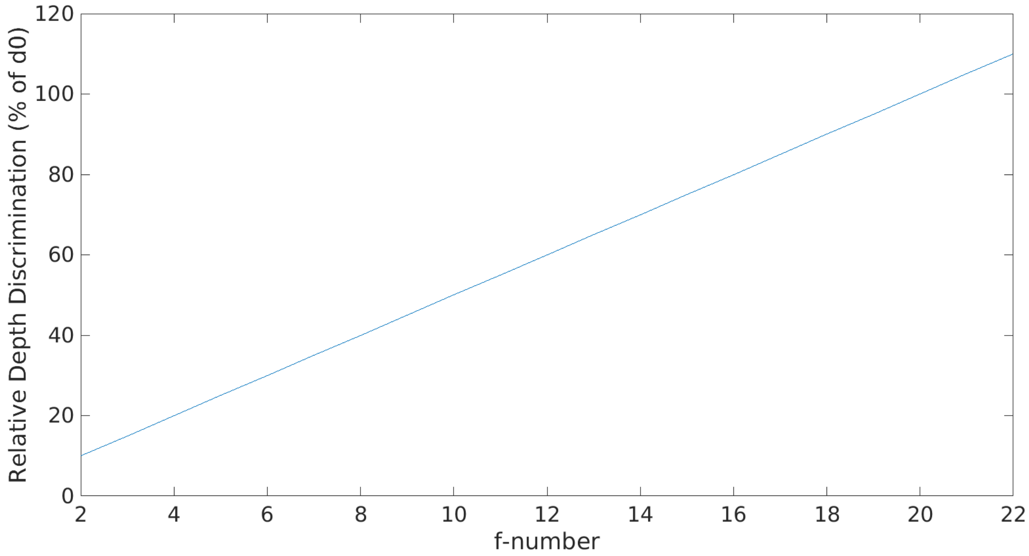
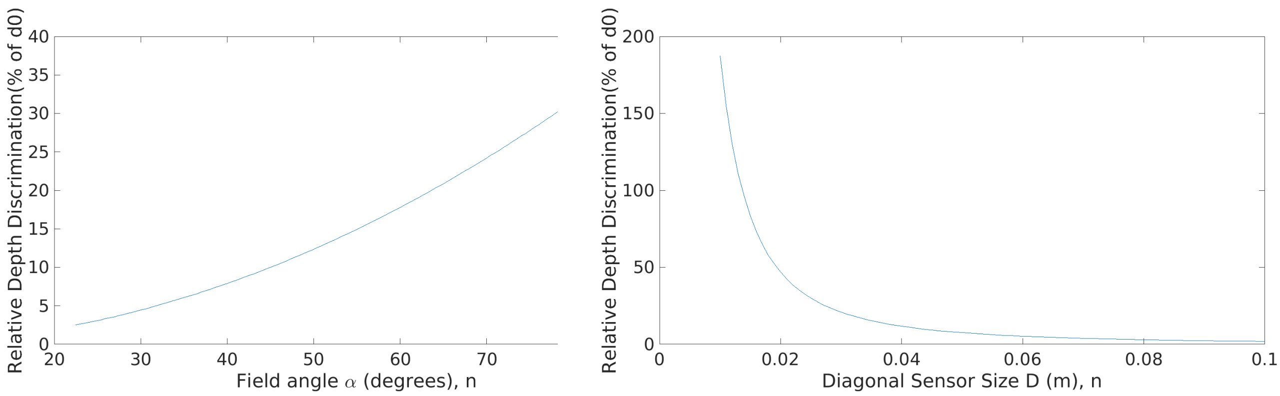


Figure 5.8: Relative depth discrimination for various f-numbers



(a) Field Angle α

(b) Sensor Size D

Figure 5.9: Two most important parameters for depth discrimination

More recently, an estimate of the depth uncertainty of water droplets has been explored by Hall [96]. In this work, the size and location of fragments produced by the impact of a water drop on a thin film of water were estimated. One of the conclusions of this paper is that not all sources of depth uncertainty are well understood, and that measured uncertainty generally follows a prediction based on the depth of focus. Other work by Hall explores depth uncertainty for refocused images [97].

5.3.2 Current Plenoptic Visual Odometry Methods

Plenoptic visual odometry has been researched for about a decade. Probably the most prolific plenoptic visual odometry researcher in literature is Donald Dansereau, who proposed three closed-form solutions for determining the six degree-of-freedom motion from a plenoptic camera [51]. In his dissertation [98], Dansereau described calibration of a plenoptic camera through a plenoptic intrinsic matrix as well as featureless 6 degree of freedom visual odometry using a gradient-based depth estimation method [99]. His work focused primarily on underwater navigation and emphasized filtering to handle in low light, murky water, and floating particles [98]. Adelson and Wang calculated the offset in pixel displacement using a least-squares gradient technique to measure the displacements [22]. The least squares estimator in the 4-D case is given in Equation (5.11)

$$h = \frac{\sum_P (I_x I_{V_x} + I_y I_{V_y})}{\sum_P (I_x^2 + I_y^2)} \tag{5.11}$$

where I_x is the spatial derivative in the x direction, I_y is the spatial derivative in the y direction, I_{v_x} is the derivative with respect to the viewing position in the x direction, and I_{v_y} is the derivative with respect to the viewing position in the y direction. Since a change in viewing position results in a corresponding change in feature position, no induced parallax exists in the orthogonal direction, as could be expected in a stereo system with nonparallel optical axes due to the differences in cameras. Dong explored a plenoptic real-time navigation system using a 3 $\times$ 3 and 40 $\times$ 30 pixel camera and showed that it provided higher accuracy than a monocular SLAM algorithm running on a 320 $\times$ 240 pixel camera [100]. The refocusing distance of a plenoptic camera was analyzed in Hahne's paper through the integral method of refocusing but was unable to provide a depth equation for the FFT-based method [101].

More recently, Zeller demonstrated a complete framework for the calibration of a plenoptic camera towards visual odometry called Direct Plenoptic Odometry [102]. Their approach resembles a monocular SLAM approach which uses subsequent frames to generate a depth map based on corresponded features from perspective images. Using bundle adjustment, their algorithm estimates a camera pose for each timestamp. In this way, they are able to avoid the pitfalls of a narrow baseline from the plenoptic camera and use what depth they are able to extract as additional information that is not available to a monocular SLAM solution. They demonstrated that their algorithm can outperform state of the art monocular SLAM approaches and is competitive with stereo approaches. Their approach does not at this time run in real-time, which is one example of the difficulty of real-time plenoptic image processing [102]. Their paper also describes a tradeoff between field of view and depth (scale), which is explored further in this dissertation.

5.3.3 Perspective Method

The perspective method for plenoptic visual odometry uses perspective images from several virtual cameras to obtain the disparity between pixels. The perspective visual odometry method implemented in this dissertation pursues a more traditional stereo odometry approach with changes made to compensate for more than two cameras as well as the rigidness of the geometry of the virtual cameras. The microlens array creates an array of small circular images (for a circular aperture) that represents the light striking each sublens. The center of these sublenses is identical to that of the pixel of a standard camera. The extra information in the circular subimages represents the different angles of light that make up the full light field [103]. The standard image (as if the camera was a traditional camera) can be recreated along with images that appear to be at various angles compared with the standard image, as shown in Figure 5.10 and Figure 5.11. With these images, the plenoptic camera becomes a virtual stereo camera.

Figure 5.10: Reconstructed left image

Figure 5.11: Reconstructed right image

Since the lightfield is not limited to two angles, many virtual cameras can be used for the navigation solution. Taking advantage of the full lightfield data is described in the next section, Section 5.3.3.1. Also worth noting is that the maximum baseline for a virtual plenoptic camera is limited by the width of the microlens size, the distance from the sensor array to the microlens array, and the nominal focal length of the lens.

5.3.3.1 Multiple Perspectives

The multiple perspective capabilities of the plenoptic camera provide a means to utilize multiple virtual cameras for extracting the disparity from a desired feature. Figure 5.12 shows four superimposed perspectives, $(u, v) = (2, 8), (8, 14), (14, 8)$, and $(8, 2)$. These perspectives are the views with the maximum baseline for the vertical view, $(8, 14)$ and $(8, 2)$, and horizontal view, $(2, 8)$ and $(14, 8)$. The four perspective views are visible from the pen in the lower left foreground. However, the table crack alongside the pen converges to a point. Additionally, the hat as well as the far right box are seemingly unaffected by the multiple perspective superimposition.

The explanation for these characteristics lies in how the virtual cameras are arranged during rendering. The perspective images rotate around a point. That point is located at the focal depth of the main lens, which is located at the hat. Perspective imaging, then, consists of a rotation around the main focal depth of the camera. Alternatively, the rotation of the virtual cameras during perspective imaging rotates around the point in focus a viewed from the raw plenoptic image, shown in Figure 5.12. These characteristics are related to Ng's description of digital refocusing as a shift and addition of the sub-aperture images (perspective images) of the light field [1]. Similarly, the reduction of integral refocusing into a perspective image as described in Section 3.4.1.1 shows the impact of reducing the refocused image down to a single perspective through removing the other perspectives by shrinking the aperture, A.

Figure 5.12: Superimposed multiple perspective images

5.3.3.2 Plenoptic Multiperspective Odometry

Multiple camera geometry has been explored for object reconstruction [104] as well as scene reconstruction, such as the famous Rome reconstruction described in [105]. This dissertation will take a different approach due to the small baseline available for the virtual cameras. In addition, no information will be stored beyond information from the previous frame in order to reduce memory requirements.

By exploiting the geometry of the virtual cameras, the disparity between features from multiple perspectives can be found to determine the depth with knowledge of the intrinsics of the camera. This disparity is the same disparity obtained through stereo odometry. However, the multiple perspectives and constant extrinsics between virtual cameras allows for a more robust solution and the ability to avoid image processing steps such as feature matchers and epipole alignment.

Figure 5.13: Raw plenoptic image from Figure 5.12

5.3.3.2.1 Virtual Camera Setup

Each perspective image is rendered from a virtual camera that is treated as a pinhole camera using the perspective rendering method directly from the raw plenoptic camera. The virtual cameras are defined as the (u, v) coordinate of the plenoptic imagery through the following equation to create a ring of virtual pinhole cameras:

$$\begin{aligned} u_p &= \frac{o}{2} + \frac{B_p}{2} cos(t) \\ v_p &= \frac{p}{2} + \frac{B_p}{2} sin(t) \end{aligned} \tag{5.12}$$

where o and p are the microlens widths in pixels, t ranges from 0 to 2π and B_p is the maximum baseline in pixels for the virtual cameras. The number of cameras desired for the ring of cameras, n_c, gives a change in t as $\frac{2\pi}{n_c}$. Given some number of desired virtual cameras results in a full ring of that number of cameras. The maximum baseline B_p is the maximum width in pixels that is feasible for perspective imaging. Since the origin of the ring of cameras lies in the center of the microlens (in (u, v) coordinates), the coefficient in front of the trigonometric terms is half of the maximum baseline width and is the true baseline from the center perspective image to the perspective images from the virtual cameras along the ring. The baseline width should be as wide as possible without losing significant portions of the image due to pixels off of the center ring. In addition to the ring of virtual cameras, the center perspective at $(8, 8)$ is chosen to create the initial features for comparison.

5.3.3.2.2 Virtual camera ring feature detection

Figure 5.14 shows feature detection using the Shi and Tomasi feature detector [73] over a $\delta t = \frac{\pi}{4}$ ring of virtual cameras and implemented through OpenCV's `goodFeaturesToTrack` function. The circles of features represent common features over the virtual cameras, while the feature in the center of the circle comes from the center perspective image at $(u, v) = (8, 8)$. Note here that some of the circles are incomplete. This is due to the fact that the feature detector for any virtual camera has no prior knowledge of the features from the other virtual cameras. In effect, all virtual cameras generate their features independently. The consistency of the features is due to the assumption that the plenoptic camera renders perspective images with similar edges and lighting so that methods such as more robust feature detetectors, feature descriptors, and feature matching are not needed. In addition, missed features in each ring is allowed since the features themselves are redundant.

The disparity for determining depth is calculated from the distance, d_f, from the center reference point to the adjoining feature from each virtual camera. Figure 5.15 shows green lines from the center perspective to each of the features from the virtual camera within a given threshold, d_{max}. Since the basis for calculating the disparity is the center perspective, this center perspective has the most important set of features. The lone red dot in the middle of the poster is an example of a feature that is exclusive to the center perspective and provides no disparity. Similarly, the green lines that do not extend in multiple directions are due to separate features that lie within the allowable d_{max}. Several of the clusters have valid feature matches except for some outliers. These situations are handled by taking the mode of the distance from the virtual camera ring to the center camera perspective feature. The disparity, then, is calculated

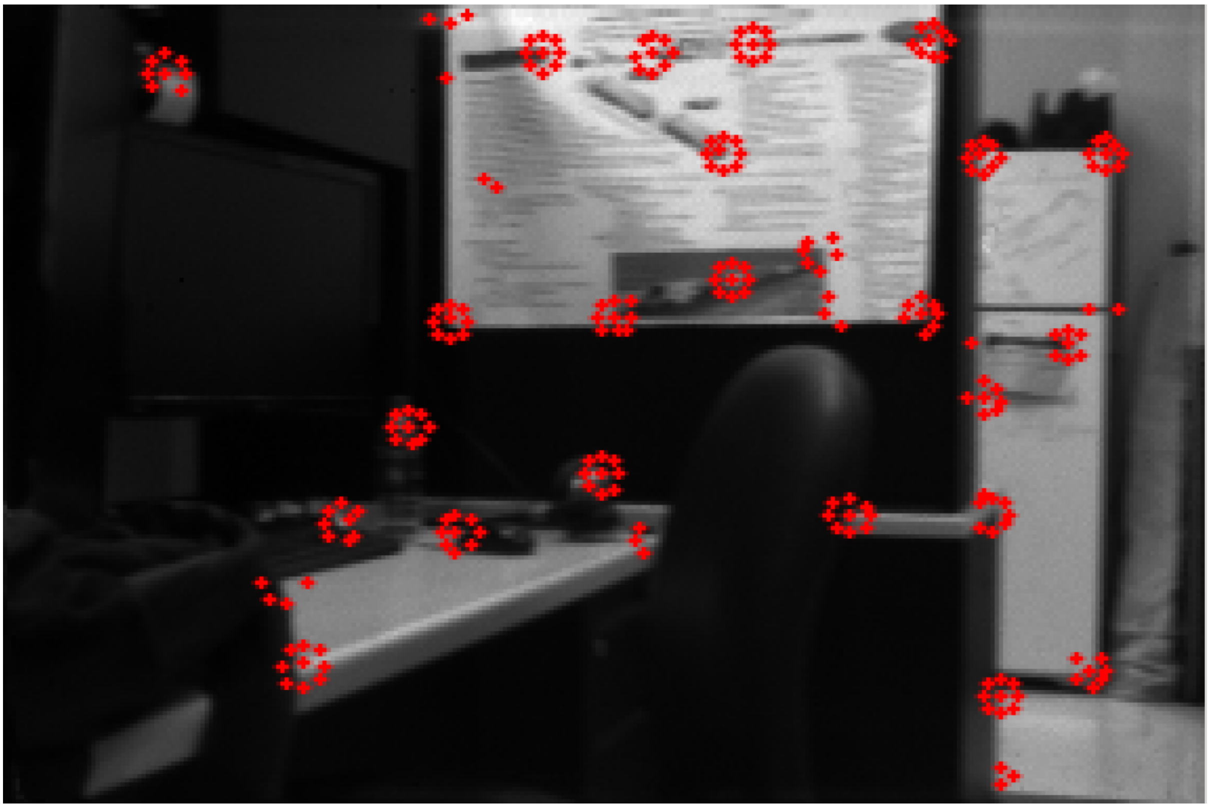


Figure 5.14: Feature detection over multiple perspectives

as shown in Equation (5.13)

$$d = 2 * Mo(\vec{d_f}); \tag{5.13}$$

where d is the disparity, $Mo()$ is the statistical mode, and $\vec{d_f}$ is a vector of the distances from the center perspective image to other features within d_{max}. Through the redundancy of the virtual camera ring, the outlier features are ignored.

5.3.3.2.3 Disparity Calculation Times

The limit of virtual cameras in the ring before dropping into subpixel rendering is based on the arc length in image coordinates as shown in Equation (5.14)

$$s_p = \frac{B_p 2\pi}{2} = 37.6 \text{ cameras} \tag{5.14}$$

where s_p is the number of pixels around the microlens at radius $\frac{B_p}{2}$ over the full circumference 2π. Approximately 37 cameras can be rendered using unique pixels. The execution time for

Figure 5.15: Perspective imaging disparity calculation

calculating the disparity using the multiperspective rendering method over 25 features is given in Table 5.1. For these execution times, the maximum baseline, B_p, was set to 12 pixels (for an effective baseline of 6 pixels) with 25 features in the center perspective image for comparison. The angle between cameras along the ring is also given in Table 5.1. These calculation times show the speed and efficiency of the perspective imaging process due to the small resolution of the rendered plenoptic images. Rendering 40 cameras and obtaining feature points for all of them takes about one fourth of a second, while rendering only eight cameras and obtaining their feature points and disparity takes approximately 50 ms.

5.3.3.2.4 Reprojection to 3D

The reprojection of the feature points (u, v) and disparity d to 3-D is the same as that used in stereo odometry. The additional virtual cameras provide redundancy, but not additional

Table 5.1: Calculation times for Generating Disparity using Multiple Perspectives

n_c	**Angle (°)**	**Time (ms)**
4	90	29.8
8	45	55.0
12	30	81.4
16	22.5	105.5
20	18	129.8
24	15	154.3
28	12.8	180.2
32	11.25	207.9
36	10	232.6
40	9	256.4

capability. The reprojection of the disparity proceeds as follows in Equation (5.15)

$$Q_R = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{1}{B} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \tag{5.15}$$

where B is the baseline of two opposing virtual cameras in the ring, f is the focal length of the virtual cameras in the ring, d is the disparity, c_x is the camera center in the horizontal camera coordinate, c_y is the camera center in the vertical camera coordinate, x is the horizontal image coordinate, and y is the vertical image coordinate. The intrinsic properties and the baseline are calculated from camera calibration as described in Section 2.3.

5.3.3.2.5 Future improvements

A clustering algorithm that classified the feature rings into separate groups could remove the need for the center perspective. Similarly, detection of the feature rings would allow for an estimate of the baseline directly. Matching algorithms can be used to guarantee feature correspondence from perspective image to perspective image. Additional processing steps, such as bundle adjustment, can reduce error as well.

5.3.4 Refocused Method

The refocusing method with plenoptic visual odometry implemented for this dissertation seeks to replace the multiple camera setup from the perspective plenoptic visual odometry with an automatic focusing algorithm to determine the depth in 3-D coordinates to a feature through the α term of the refocusing algorithm. This section describes a depth-from-focus algorithm to extract the depth of the feature.

5.3.4.1 Autofocus

The autofocus algorithm determines the value of α that corresponds to a feature when it is in focus. The fundamental function of what determines if an image is "in focus" has been researched primarily for consumer digital cameras [106]. These approaches include eliciting infrared or ultrasound using depth sensors [107], using stereo camera depth information to refocus the camera [108], gray-level measurements for a given set of focus [109], contrast detection [110], and the use of phase detection [111]. A significant departure from most of these techniques with the plenoptic camera is that the refocusing for a plenoptic camera is software-oriented, while in these applications the main lens is moved directly. The approach for autofocusing in this dissertation is to compare the refocused image with a full depth of field image (a perspective image) to determine if a feature is in focus.

5.3.4.1.1 Correlation Function, f_c

Fortunately for plenoptic imaging, the perspective image provides a focused image with which to compare the refocused image. An autocorrelation function through template matching is used to compare a given patch of refocused image to the focused patch of the perspective image. This function shown below was described by Lewis [112], and its computational efficiency was analyzed in [113]

$$R(s,t) = \sum_{s'=0}^{m_t} \sum_{t'=0}^{n_t} T'(s,t') I'(s+s', t+s') \tag{5.16}$$

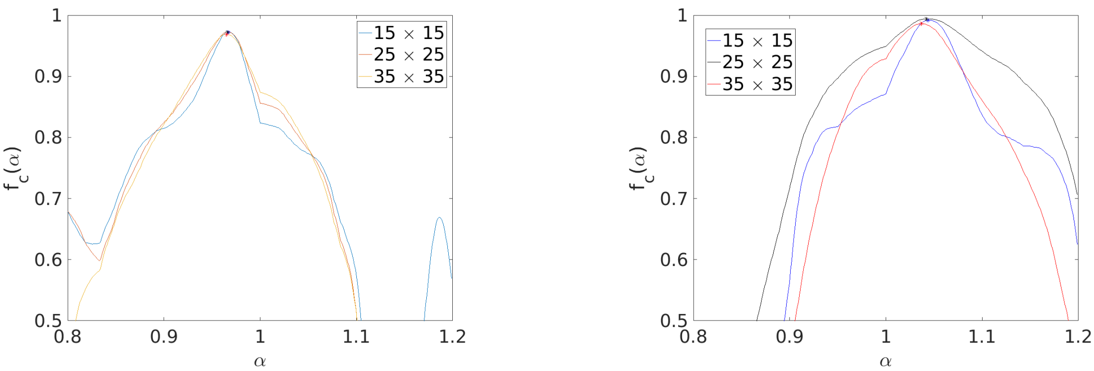
where

$$
\begin{aligned}
T'(s',t') &= T(s',t') - \frac{1}{m_t n_t}\sum_{s''=0}^{m_t}\sum_{t''=0}^{n_t} T(s'',y'') \\
I'(s+s',t+t') &= I(s+s',t+t') - \frac{1}{m_t n_t}\sum_{s''=0}^{m_t}\sum_{t''=0}^{n_t} I(s+s',t+t')
\end{aligned}
\tag{5.17}
$$

The implementation of template matching uses the OpenCV function `matchTemplate`. Since the location of the template is known in the image, the template match only needs to cover the same area as the template on the refocused image. A small buffer consisting of a slightly larger area than the template can provide some flexibility for the template matcher at the cost of some additional computational time. The max value of the correlation in the patch can simply be found using Equation (5.18).

$$
f_c(\alpha) = max(R). \tag{5.18}
$$

Figure 5.16 shows the correlation curves for the area around the face and the clock in Figure 5.12 over a range of α for template sizes of 15×15, 25×25, and 35×35. Note that the peak for the face is significantly narrower over the three template sizes than the clock target due to its more complex characteristics.



(a) Face target correlation curves

(b) Clock target correlation curves

Figure 5.16: Correlation curves from template matching for the face and clock targets

Edge Cases

The face target and clock target are ideal for measuring how in-focus a template patch is since each feature is close to the same depth. For the cases in which a template contains both near features and far features, the result is much more ambiguous, as shown in Figure 5.17.

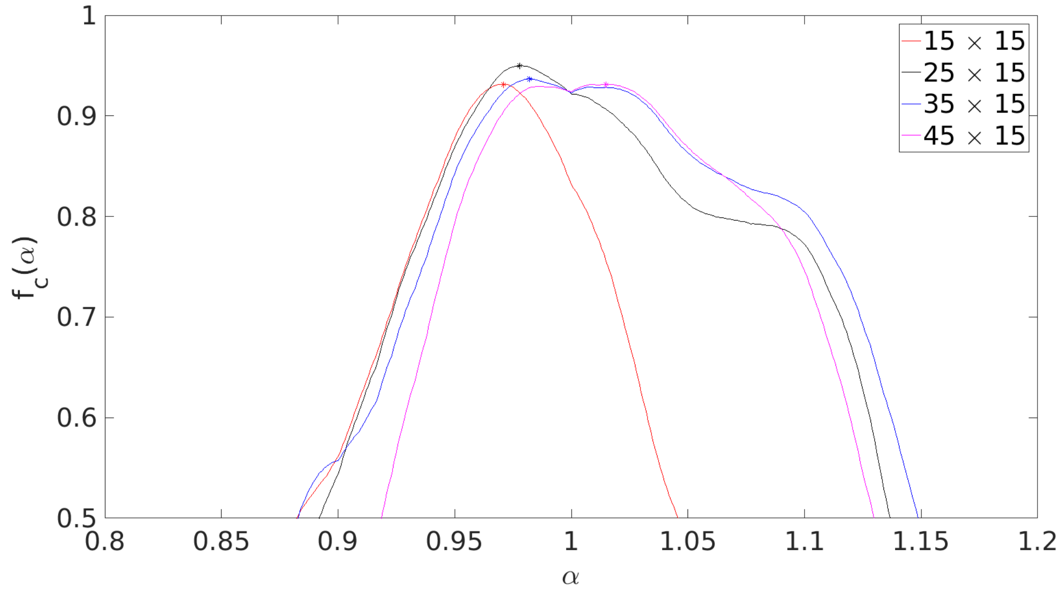


Figure 5.17: Edge target correlation sizes

Figure 5.18 shows the target sizes of the templates used for generating the correlation curves of Figure 5.17. The smallest target, which consists of the red square and curve, clearly shows a strong maximum at $\alpha = 0.93$, which focuses on the background. The black rectangle, which consists of the red square and the additional 10 pixels outlines in black, has a more ambiguous curve since a portion of the target template is on the edge of the clock. The blue rectangle, which consists of a larger portion of the clock, has another ambiguous curve and the maximum correlation is lower than that of the black template. Finally, the magenta curve is very ambiguous as seen in Figure 5.17. The maximum for the magenta curve is only slightly higher for focusing nearby at $\alpha = 1.01$. The magenta template consists of approximately a third of the scene behind the clock, while the other two-thirds consist of the clock. However, the features of the clock portion of the magenta template are relatively benign with vertical edges, while the scene behind the clock is much more complex. The additional complexity of the scene in the back pulls the correlation curves higher even though its area in the magenta template is only half that of the clock portion. Note that the $\alpha = 1$ case, while continuous over the correlation, creates a significant elbow in the curves.

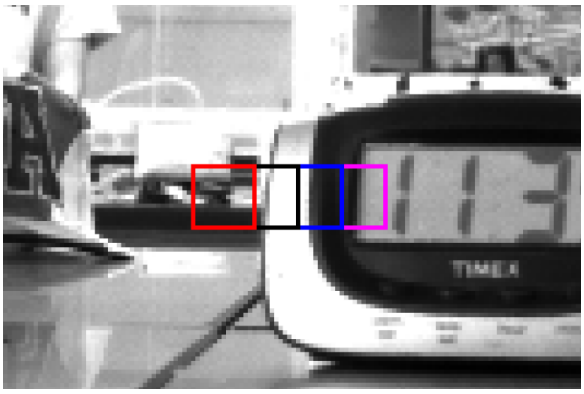


Figure 5.18: Edge target sizes

5.3.4.1.2 Golden Section Search

Golden section search [114] was chosen to find the best value of α determined through the output of template matching for a range of α. The golden section search algorithm exploits the symmetry of the golden ratio to gradually decrease the possible range of values within which is the desired α that gives a correlation within a given tolerance, τ.

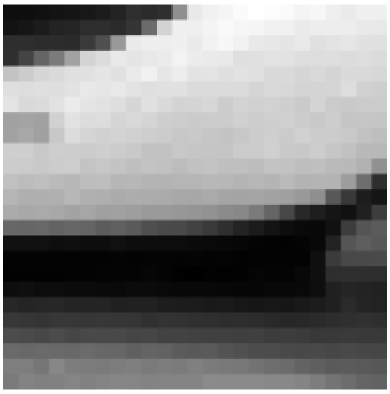
Algorithm 2 shows the golden section search as implemented in this dissertation for autorefocusing. a and c contain the maximum and minimum range within which lies the desired α. b is the center point of the algorithm, and $f(x_2)$ is the precomputed correlation value (since it does not change) at point x_2. τ is the tolerance for ending the iterative process. I_p is the perspective sub image used as the template for template matching. In Algorithm 2, $\phi = \frac{1+\sqrt{5}}{2} = 1.618$ and $\phi_{res} = 2 - \phi$. α_1 and α_2 are the previous α values, and f_{c1}, f_{c2} are the previous correlation outputs at the corresponding α values. This algorithm recursively searches for the α value that maximizes the correlation outputs within a tolerance. Since the template matching algorithm can result in values that are not strictly unimodal for small changes in α, the inputs for a and c are very wide, while b is initially set to $\frac{\phi}{2}$.

Algorithm 2 Golden Section Search

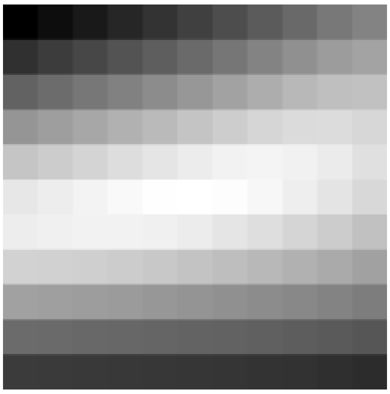
```
procedure GoldenSectionSearch(a,b,c,τ,f(x_2),I_p)
    if c − b > b − a then
        x = b + φ_res(c − b)
    else
        x = b − φ_res(b − a)
    end if
    if |c − a| < τ(|b| + |x|) then                                ▷ Check tolerance
        return (a+c)/2                                            ▷ If below tolerance, end
    end if
    if α_1 ≡ b then
        f_b = f_c1
    else
        if α_2 ≡ b then
            f_b = f_c2
        else
            f_b = f(b, I_p)
        end if
    end if
    if α_1 ≡ x then
        f_x = f_c1
    else
        if α_2 ≡ x then
            f_b = f_c2
        else
            f_b = f(x, I_p)
        end if
    end if
    α_1 = b,α_2 = x,f_c1 = f_x,f_c2 = f_b                         ▷ Update previous values
    if f_x > f_b then
        if c − b > b − a then
            return GoldenSectionSearch(b,x,c,τ,f(x_2),I_p)        ▷ Recursive Call
        else
            return GoldenSectionSearch(a,x,b,τ,f(x_2),I_p)
        end if
    else
        if c − b > b − a then
            return GoldenSectionSearch(a,b,x,τ,f(x_2),I_p)
        else
            return GoldenSectionSearch(x,b,c,τ,f(x_2),I_p)
        end if
    end if

end procedure
```

5.3.4.1.3 Autofocus Examples

Figure 5.19 shows the result of the autofocus algorithm on a template consisting of the lower edge of a clock. The autocorrelation output shows a band of high likelihood areas, which correspond to the edges seen in the template. Templates such as these that have repeating or similar characteristics can make identifying the optimum template match difficult, but in this case the correct location was identified. Note also that the buffer around the template can be reduced to 0, which would result in a single autocorrelation output and increased computational efficiency since the other autocorrelation numbers are not calculated.



(a) Template



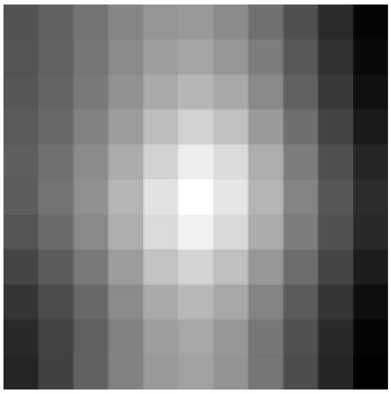
(b) Autocorrelation output

(c) Original image with template (blue), search area (green), and matched location (red)

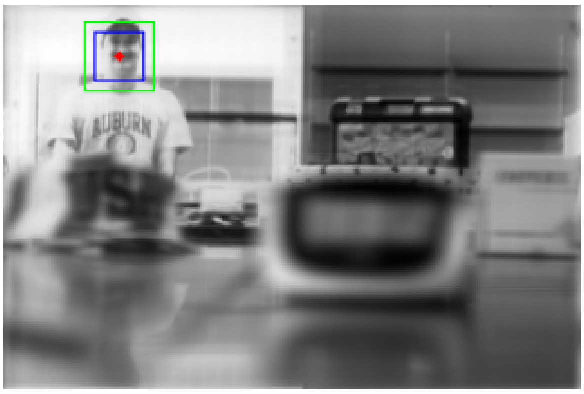
Figure 5.19: Autofocus algorithm - autofocus on clock corner

Figure 5.20 shows a more complex template of a man's face. Its autocorrelation result is significantly less ambiguous with a peak in the center, as seen in Figure 5.20b. Figure 5.20c shows the result of the autofocus algorithm for the face template. In contrast to Figure 5.19, the man is in focus, while the clock is out of focus.

(a) Template



(b) Autocorrelation output



(c) Original image with template (blue), search area (green), and matched location (red)

Figure 5.20: Autofocus algorithm - Autofocus on Face

Example Autofocus Times

Execution times for autofocus are dependent on the tolerance and the time to calculate the correlation function, f_c, due to the iterative nature of the algorithm. More accurate autofocus results requires more iterations, which in turn results in a greater number of calculations of the correlation function. Table 5.2 shows the execution time for the autofocusing shown in Figure 5.19. Since the range correlation execution times are similar given the same width and height for the sub image, an example of the computation times for the range correlation is given for the precalculated $f_c(x_2)$. In this particular example, the golden search required ten correlation execution times for finding the α with error within the tolerance. The execution time for the golden search algorithm is about 10 times the execution time for $f_c(x_2)$, with some additional computation necessary for the recursive calls as well as updating α_1, α_2, f_{c1}, and f_{c2}. The template size is 25×25, which is large for determining the correlation, but in practice smaller template sizes result in failed autofocusing. In addition, the tolerance for this example was set to 0.1, which is high but enough to focus on a feature.

Note in Table 5.2 that the perspective image calculation is significantly less than that presented in Table 4.15. For this implementation, the perspective image was calculated directly from the radiance used in refocusing with a data type of `fftw_complex` rather than `double`. As such, the transport time to the GPU and time for generating the radiance do not apply for generating this perspective image. In addition, the perspective image was only calculated over the sub image, which is significantly smaller than the entire $m \times n$ perspective image. The resulting execution time, then, is significantly smaller for this particular application.

Table 5.2: CUDA Rendering for FFT-based Autofocusing

Autofocus Timing		**(ms)**
Perspective Sub Image		0.065
Range Correlation - $f_c(x_2)$		
Slice	1.814	
Refocused Sub Image	.001	
Match Template	.023	
Find Max Correlation	.001	
Total		1.839
Golden Search (10 correlations)		20.274
Total		22.178

With the total time to refocus a feature at 20.25 ms, a real-time application for calculating a point cloud becomes impossible. With 100 features, the time required to calculate the depth to each feature is 2 seconds and exceeds the allocated time given for real-time performance at . A separate method, then, is needed for calculating the depth for many features.

5.3.4.2 Sweep Method

The sweep method iterates over the stack of refocused images and conducts template matching over all of the features for each α-refocused image. Each feature has a corresponding maximum α_m value, which represents the current maximum α term that results in the most in-focus image. The size of the refocused stack, which is chosen through the change in α over a valid α range, and the number of features in an image can be used to design the refocusing algorithm to meet a maximum computational time requirement. After iterating through the focal stack, the α_m term for each feature can be used as an input to the chosen ranging model equation, which is discussed in the next section.

5.3.4.3 Depth through Refocusing Calibration

For the plenoptic camera, the depth vs. disparity plot is shown in Figure 5.21. Note that the curve is hyperbolic, and the depth is increasingly difficult to discern from disparity at far distances. Unfortunately, this range equation requires that the image planes between the stereo

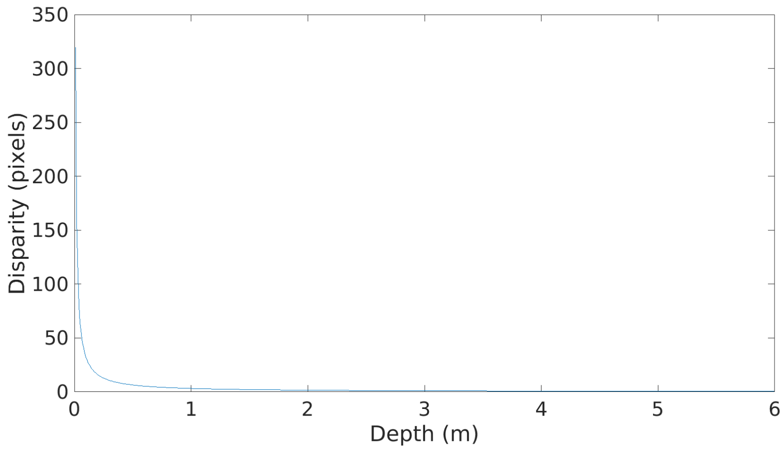


Figure 5.21: Hyperbolic equation of disparity vs. depth

cameras are perfectly rectified. Additionally, calculating perfectly rectified stereo cameras is difficult. The solution for plenoptic imagery, then, is to use known distances in calibration to create a function to relate the in-focus α from the autofocus or α sweep method to distance. Like α, larger disparities result in closer features, while smaller disparities result in far features. Assuming some proportional relationship between disparity and α, the relationship between α and distance can be modeled as hyperbolic.

$$Z = \frac{c}{\alpha^b} \tag{5.19}$$

where Z is the distance in meters, α is the coefficient for focal length for synthetic imaging, and c and b are desired terms based on experimental data. This experimental data is obtained by setting a target at a known distance from the camera, and determining what α value corresponds to the point at which the sign is most in focus. At $\alpha = 1$, the synthetic sensor array is the actual sensor array, and the image is much sharper at this α since no interpolation is needed. Care must be taken in determining a model for α and range.

Five non-linear models (four hyperbolic models and an exponential model) were fit to the maximum correlation at α at a known range. The next sections describe the models and the equations for their fit, while the next chapter shows the experimental data and the quality of the fits for the plenoptic camera.

5.3.4.3.1 Hyperbolic Model - $\frac{c}{\alpha^b}$

One hyperbolic model used in the range calibration fit is shown in Equation (5.20) and is based on the hyperbolic model of Equation (5.19).

$$Z = \frac{c}{\alpha^b} \tag{5.20}$$

where c and b are the estimated model parameters, Z is the known range, and α is the maximum correlation calculated from focal refocusing.

Calculation of b_0, c_0

The hyperbolic model is linearized through a logarithmic approach as shown in Equation (5.21)

$$\log(\vec{Z}) = \log \frac{c}{\vec{\alpha}^b} \tag{5.21}$$

where $\vec{\alpha}$ and $\vec{Z}$ are vectors of α and distances Z. Through basic logarithmic algebra, the following equation is derived.

$$\log(\vec{Z}) = \log c - b \log(\vec{\alpha}) \tag{5.22}$$

Equation (5.22) is moved into matrix form.

$$\log(\vec{Z}) = \begin{bmatrix} 1 & -\log(\vec{\alpha}) \end{bmatrix} \begin{bmatrix} \log c \\ b \end{bmatrix} \tag{5.23}$$

Rearranging the equation to find the unknowns results in Equation (5.24)

$$\begin{bmatrix} \log c \\ b \end{bmatrix} = \begin{bmatrix} 1 & -\log(\vec{\alpha}) \end{bmatrix}^{-1} \log(\vec{Z}) \tag{5.24}$$

where the data collected from experimentation can be used to find c and b.

5.3.4.3.2 Hyperbolic Model with offset - $a + \frac{c}{\alpha^b}$

The data may not have a horizontal asymptote of zero. Therefore, the addition of an offset in the model can help better capture the curve. The hyperbolic model used in the range calibration fit is shown in Equation (5.25) and is again based on the hyperbolic model of Equation (5.19).

$$Z = a + \frac{c}{\alpha^b} \tag{5.25}$$

where a, b, and c are the estimated model parameters, Z is the known range, and α is the maximum correlation calculated from focal refocusing. The initial variables are calculated as described in Section 5.3.4.3.1 with the addition of a given a_0.

5.3.4.3.3 Hyperbolic Model with horizontal and vertical offset $a + \frac{c}{(\alpha+d)^b}$

The data may not have a horizontal asymptote of zero or a vertical offset of 0. The addition of both of these offsets in the model can help better capture the curve for such data. The hyperbolic model used in the range calibration fit is shown in Equation (5.26)

$$Z = a + \frac{c}{(\alpha + d)^b} \tag{5.26}$$

where a, b, c, and d are the estimated model parameters, Z is the known range, and α is the maximum correlation calculated from focal refocusing.

5.3.4.3.4 Michaelis-Menten Hyperbolic Model $\frac{V_{min}\alpha}{K_M+\alpha}$

Another candidate for the α to range mapping is another hyperbolic model with a different parameterization, shown in Equation (5.27), known as the Michaelis-Menten model [115].

$$Z = \frac{V_{min}\alpha}{K_M + \alpha} \tag{5.27}$$

This model is often used in enzyme kinetics to describe the reaction rate of an enzyme to the concentration of a substrate based on V_{max}, the maximum rate of the system, and K_M, which is half of V_{max}. For this dissertation, the range is related to α based on these same two parameters, where V_{min} is the horizontal asymptote corresponding to the minimum range to the object, and K_m corresponds to twice V_{min}.

V_{min0} and K_{M0} Calculation

The Michaelis-Menten model, while linear, can be transformed into an intrinsically linear form by taking the reciprocal of each side of the equation.

$$\frac{1}{Z} = \frac{K_M + \alpha}{V_{min}\alpha} = \frac{K_M}{V_{min}\alpha} + \frac{1}{V_{min}} = \frac{K_M}{V_{min}}\frac{1}{\alpha} + \frac{1}{V_{min}} = A\frac{1}{\alpha} + B \tag{5.28}$$

This equation is known as the Lineweaver-Burk equation [116] and converts the non-linear Michaelis-Menten equation into an intrinsically linear equation, as seen in Figure 5.22. Determining the values of A and B is straightforward, and thus $V_{min0} = \frac{1}{B}$ and $K_{m0} = \frac{A}{B}$. This approximation works well for regression problems with small errors, but the reciprocal increases any error. As such, the Lineweaver-Burk equation is used as the initial condition for the nonlinear regression.

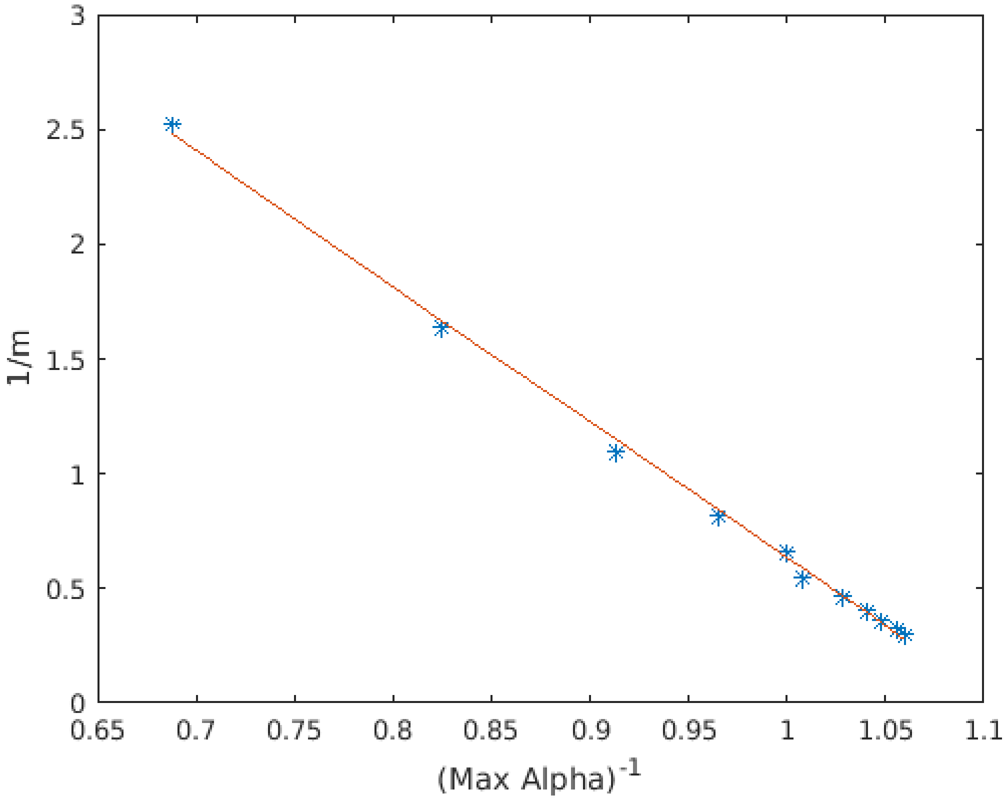


Figure 5.22: Lineweaver-Burk plot and linear fit for intrinsically linear Michaelis-Menten equation

5.3.4.3.5 Michaelis-Menten Hyperbolic Model with Offset

Another Michaelis-Menten for the α to range mapping adds a horizontal offset with a different parameterization, shown in Equation (5.29).

$$Z = \frac{V_{min}(\alpha + d)}{K_M + (\alpha + d)} \tag{5.29}$$

The horizontal offset shifts the vertical asymptote. For this dissertation, the range is related to α based on these same two parameters, where V_{min} is the horizontal asymptote corresponding to the minimum range to the object, and K_m corresponds to the α value at twice V_{min}. The initial condition is set similarly as the Michaelis-Menten model as well as a small vertical offset for the d_0 parameter.

5.3.4.3.6 Exponential Model

An exponential model, while not adhering to the standard range equation of Equation (5.21), is another candidate for fitting the range and α data for range calibration and is shown in Equation (5.30).

$$Z = a + be^{c\alpha} \tag{5.30}$$

Calculating b_0, and c_0

Two of the exponential model initial conditions, b_0 and c_0, can be approximated by a least squares solution of the exponential equation in Equation (5.31).

$$Z = Be^{C\alpha} \tag{5.31}$$

This least squares solution can be found by minimizing

$$\sum_{i=1}^{n} y_i(\log y_i - a - bx_i)^2 \tag{5.32}$$

and

$$\begin{aligned} a &= \frac{\sum_{i=0}^{n}(x_i^2 y_i)\sum_{i=0}^{n}(y_i \log(y_i)) - \sum_{i=0}^{n} x_i y_i \sum_{i=0}^{n}(x_i y_i \log(y_i))}{} \\ b &= \frac{n\sum_{i=0}^{n}(x_i \log(y_i)) - \sum_{i=0}^{n} x_i \sum_{i=0}^{n} \log(y_i)}{n\sum_{i=0}^{n} x_i^2 - (\sum_{i=0}^{n} x_i)^2} \end{aligned} \tag{5.33}$$

where $B = b$ and $A = e^a$ [117]. The initial condition for horizontal asymptote, a_0, is approximated from the data.

5.3.4.3.7 Non-linear fit for all models

The non-linear regression for each model is straightforward. MATLAB's `fminsearch` algorithm was used for the non-linear regression [118]. For each model Z_m and parameter list

p, the minimization equation is described in Equation (5.34).

$$\sum_{i=0}^{n}(Z_i - Z_m(p, \alpha_i))^2 = 0 \tag{5.34}$$

5.3.4.3.8 Fit Metrics

Evaluation of each model is based on a series of fit metrics along with visual comparison with the original data. The sum of squares of error, or the summed square of residuals, is given by

$$SSE = \sum_{i=0}^{n}(Z_i - \hat{Z}_i)^2 \tag{5.35}$$

and a small SSE indicates a good regression model. The sum of squares of the regression, SSR, is

$$SSR = \sum_{i=0}^{n}(Z_i - \hat{Z}_i)^2 \tag{5.36}$$

where sum of squares about the mean, SST, is defined below.

$$SST = \sum_{i=0}^{n}(Z_i - \hat{Z}_i)^2 \tag{5.37}$$

The R-square statistic measures the quality of the fit based on the variation of the data.

$$R^2 = 1 - \frac{SSE}{SST} \tag{5.38}$$

Its value is typically between 0 and 1, where 1 indicates that the model perfectly describes the data, although negative values are possible when the intercept point is not specified. Typically, a negative R-square statistic will result in a bad fit. The root mean squared error and the mean squared error are

$$\begin{aligned} MSE &= \frac{SSE}{v} \\ RMSE &= \sqrt{\frac{SSE}{v}} \end{aligned} \tag{5.39}$$

where v is the residual degrees of freedom, or $v = n - m$. A smaller $RMSE$ indicates a better regression model. The performance of each models based on these statistics is described in Section 6.5.

5.3.4.3.9 Reprojection to 3-D

Instead of the range model based on the disparity, $Z = \frac{Bf}{d}$, the range model provides a direct function to obtain the range Z from the refocused image at α. The Q matrix used to reproject each point to 3-D must be slightly changed. Equation (5.40) shows the Q matrix using the range equation without the disparity and the baseline.

$$\begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{1}{B} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x + c_x \\ y + c_y \\ f \\ \frac{d}{B} \end{bmatrix} = \begin{bmatrix} \frac{B(x+c_x)}{d} \\ \frac{B(y+c_y)}{d} \\ \frac{Bf}{d} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{Z(x+c_x)}{f} \\ \frac{Z(y+c_y)}{f} \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x + c_x \\ y + c_y \\ f \\ \frac{f}{Z} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & f & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \frac{1}{Z} \\ 1 \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ \frac{1}{Z} \\ 1 \end{bmatrix} \tag{5.40}$$

5.4 Summary

This chapter has described several visual odometry algorithms, including a traditional stereo camera algorithm that uses sparse point clouds to determine the rotation and translation of a camera from one frame to the next frame. One of the most important tasks in using 2-D images to measure feature locations in a 3-D world is ranging. Three ranging methods with a plenoptic camera were described: a virtual stereo camera approach, a multiperspective approach, and a refocused rendering approach. The next chapter will focus on each of these algorithms as implemented on a real camera for several different sizes of lens.

Chapter 6

Experimental Results of Plenoptic Point Cloud Generation

Experimental data was collected on a custom built plenoptic camera developed by Auburn University's Advanced Flow Diagnostics Laboratory (AFDL). Three datasets were taken for calibration: microlens calibration (Section 3.1), ranging calibration (Section 6.5), and stereo calibration (Section 6.3). This chapter begins with an overview of the experimental plenoptic camera and test environment.

6.1 Test Setup

The plenoptic camera used for the experimental results in this dissertation was an ICL-B4820 IMPERX camera [119]. The maximum frame rate for this camera is 4.3 frames per second using the full 4904$\times$3280 resolution of the camera but can be increased with lower resolution. The bit depth was 14 bits, which is smaller than the 16 bit recorded raw imagery. The camera was monochrome, which results in less computation time. The pixel size of the camera was 7.40 μm. The camera was connected through a CameraLink interface [120] to a frame grabber, which in turn was connected through an ExpressCard slot on a laptop with a maximum bandwidth of 280 megabits per second [121]. The exposure setting was set to an internal setting that provided an adequate contrast in intensity, but no auto exposure was enabled. Note that this camera was built specifically for measuring distances very close to the camera. This dissertation explores ranges far beyond for which it was designed.

As before, the microlens were projected onto a 16$\times$16 pixel patch. The microlens calibration algorithm searched for a 286$\times$190 microlens array for creating the radiance. The f-number for the 135mm data was 4, and with a 135mm focal length, the effective aperture width is 33.8mm. Note that while vertical virtual cameras are possible, the maximum baseline in the vertical direction is limited by the vertical resolution of the camera.

Three datasets were collected with three different lenses with the plenoptic camera: 55mm, 80mm, and 135mm. For the 55mm lens, the focus was set to infinity. For the 135mm lens, the focus was set to 4 feet, or 1.2 meters. The fstop for each lens was decreased until the microlens circles were barely touching in the raw plenoptic image.

Several dead pixels were present in the imagery and required some additional image processing for removal. These pixels were detected as features and tracked as non-moving features through motion imagery. These dead pixels resulted in significant problems for the tracking algorithm until a small blurring option removed the dead pixels. Alternatively, camera control software can remove these dead pixels as well.

The plenoptic camera was mounted on a robot from Auburn University's GPS and Vehicle Dynamics Laboratory (GAVLAB) pointing forward, as seen in Figure 6.1. The calibration board was held in various positions in front of the camera, and the ranging target setup in front of the camera and moved to each known distance. The microlens calibration dataset was collected by setting a brightly lit white board in front of the camera to ensure bright pinhole points at the center of each microlens.

All tests were conducted in Ubuntu version 14.04 on an Intel Core i7-4720HQ CPU with 4 cores (8 with multithreading), a GeForce GTX 980M graphics card, a 1 TB spinning disk hard drive, and 16 gigabytes of Random Access Memory (RAM). The number of other running processes was minimized.

Figure 6.1: IMPERX Plenoptic camera mounted to ATRV with 135mm lens

6.2 Challenges of the Plenoptic Camera for Navigation

Any camera used for navigation will benefit from high resolution, large field of view, and a high imagery rate. Each of these characteristics adds additional information available for pulling information from a scene for navigation. Higher frame rates allow for more captured motion, which leads to more dynamic movements of the camera. The experimental plenoptic camera has a relatively low frame rate (for a vision algorithm) of 4-5 hz for a 4904 $\times$ 3280 image [119]. Any motion detected by the plenoptic camera should be relatively slow to capture the dynamics of the scene.

6.2.1 Resolution

A major problem with the experimental plenoptic camera used in this dissertation as a navigation sensor is the small resolution of the rendered images. Figure 6.2 shows the relative sizes of the plenoptic image and the rendered image. The majority of the information contained inside of the plenoptic image consists of the various angular light field data. Current commercial plenoptic cameras have much higher resolutions, such as the Lytro Illum with a maximum resolution of 2450 $\times$ 1634 [122]. However, even this larger resolution is smaller than other

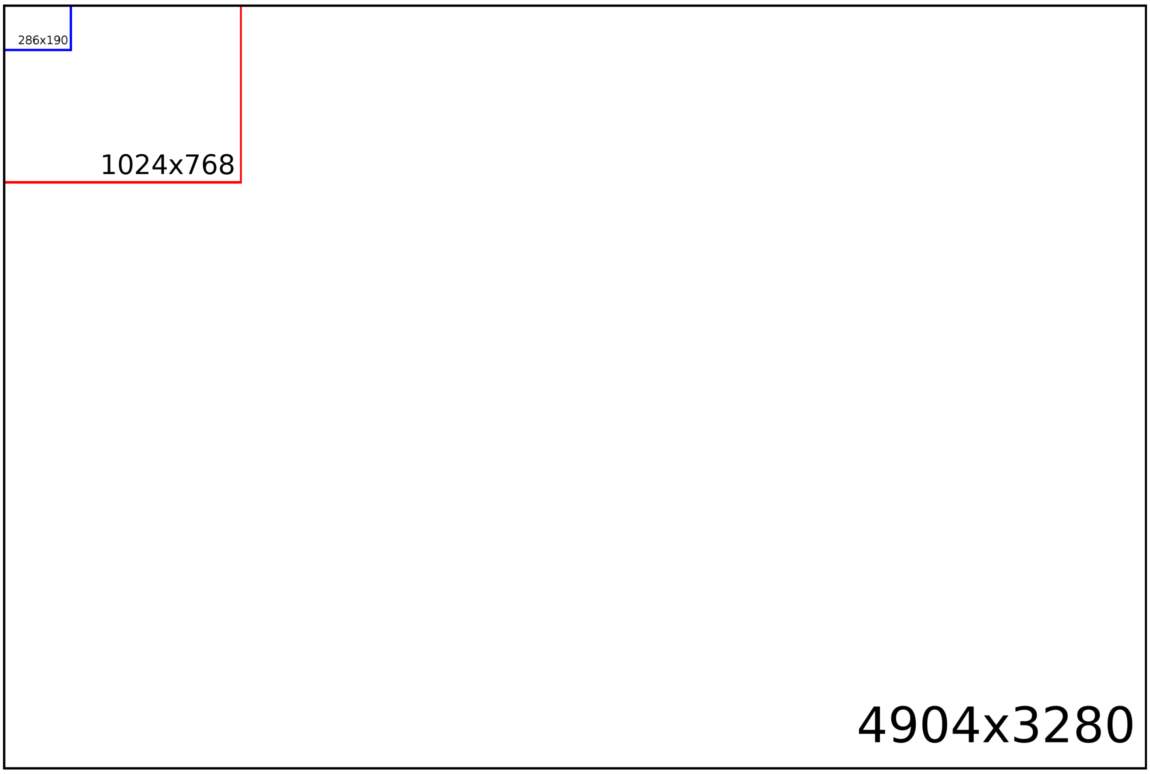


Figure 6.2: Resolution of the plenoptic image (black), standard display resolution (red), and the rendered image (blue)

commercial cameras since the angular information must be saved in the raw camera image. A low resolution image is effectively a low visual sample of a scene. This low sampling results in less information for extracting features.

In addition, the 4-D radiance for the experimental camera uses only about 79% of the available pixels. The other 21% are black pixels from light striking between the microlens, as seen in Figure 3.1 of Chapter 3. The plenoptic image itself can have unusable rows and columns of microlens due to the angular offset of the microlens array, which reduces the usable resolution of the rendered images.

6.2.2 Field of View vs. Baseline

The maximum baseline for a plenoptic camera is a function of the focal length of the main lens f, the distance from the microlens array to the sensor array d_m, and the diameter w_m as seen in Equation (6.1) [123].

$$B_{max} = \frac{w_m f}{d_m} \tag{6.1}$$

For a plenoptic camera with a distance of w_m =300 μm, a main lens focal length of 135mm, and a microlens diameter of .12mm, the maximum baseline is 53.3mm. This maximum baseline does not take into account any edge shadows that arise at the border of the microlens and assumes perfect image renderings at the maximum width of the microlens. The actual maximum baseline, then, is smaller.

The field of view of a camera can be calculated through the mechanical focal length and the sensor size in real world coordinates is given in Equation (6.2)

$$FOV = 2\tan^{-1}(\frac{d}{2f_w}) \tag{6.2}$$

where d is the dimension and f_w is the focal length in real world coordinates. Alternatively, the calibrated focal length and image size in pixels can be used to determine the field of view. Generally, the calibrated focal length is more accurate since manufacturing differences are compensated for in the calibrated focal length. Using pixel coordinates, the field of view is given in Equation (6.3)

$$FOV = 2\tan^{-1}(\frac{d_{wh}}{2f}) \tag{6.3}$$

where f is the focal length in pixels and d_{wh} is the desired dimension size (either width or height) in pixels. The field of view of each lens for a 286×190 perspective image as well as the baseline found through the extrinsic calibration is given in Table 6.1.

Table 6.1: Field of View and Baseline for Three Lenses

Lens (mm)	f_x **(pixels)**	f_y **(pixels)**	**HFOV (°)**	**VFOV(°)**	**B(mm)**
55	386.8	387.9	40.57	27.5	8.28
80	696.8	698.1	23.19	15.50	18.67
135	1014.1	1014.3	16.05	10.70	27.39

The images using the three lenses of Table 6.1 are shown in Figure 6.3 at a distance of four feet. The 55mm lens image shows almost the entire sign, while the 135mm lens image rendering results in a smaller field of view due to the zoomed image. Since the 135mm lens provides greater range capability than the other lenses, a tradeoff exists between field of view and range capability. A wide field of view is important for ensuring features are consistent from frame to frame. A tradeoff, then, exists between depth capability and feature tracking ability.

(a) Perspective image at 4 feet with 55mm lens

(b) Perspective image at 4 feet with 80mm lens

(c) Perspective image at 4 feet with 135mm lens

Figure 6.3: Perspective images for 3 different lenses

6.3 Stereo Calibration

Plenoptic images were taken of a calibration board, and the intrinsics and extrinsics of the virtual cameras for $(u_1, v_1) = (2, 8)$ and $(u_2, v_2) = (13.5, 8)$ were estimated. Note that this pair of virtual cameras are oriented horizontally. In addition, the proximity of the perspectives to the edge of the visible microlens results in some darkening at the edges. Narrower perspectives were not selected in order to maximize the baseline of the virtual cameras.

6.3.1 Intrinsic Calibration

As mentioned in Section 2.3.1, the intrinsic parameters consist of the focal lengths of the camera, f_x and f_y, the image centers, c_x and c_y, and finally the distortion parameters. The intrinsic calibration was set to keep the camera center at half of the width and height of the image and to estimate three radial distortion parameters and two tangential parameters. The camera matrices for the left and right virtual cameras are given below.

$$M_L = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1014.1 & 0 & 142.5 \\ 0 & 1015.0 & 94.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

$$M_R = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1014.4 & 0 & 142.5 \\ 0 & 1013.6 & 94.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

Note that the focal lengths of both of the virtual cameras are similar. The distortion parameters are shown in Table 6.2. The distortion parameters of the left and right virtual cameras are shown in Table 6.3 where p_1 and p_2 are the tangential distortion coefficients and k_{1-3} are the

Table 6.2: Distortion Parameters from Left Image from 135mm lens

Parameter	**Distortion**
k_1	-.24756
k_2	-35.941
k_3	923.83
p_1	-.00088
p_2	.00542

Table 6.3: Distortion Parameters from Right Image from 135mm lens

Parameter	**Distortion**
k_1	-.17527
k_2	-29.100
k_3	670.36
p_1	-.00115
p_2	-.00563

radial distortion coefficients. Due to the high quality components of this camera, the distortion parameters were restricted to these two tangential radial components and three radial distortion components. The distortion parameters are small as seen in Figure 6.4. In this figure, the edges of the images have been shifted as part of the rectification process, but little warping was needed to compensate for distortion. The focal lengths for both virtual cameras and the x, y directions are almost identical, which is encouraging given that the focal lengths are expected to be the same for the virtual plenoptic cameras. The principal point was fixed to the center of the image. Figure 6.4 also shows a border (red) of the calibrated portions of the images. Note in this figure that the green lines in one image bisect the corresponding features in the other image. This observation is needed to ensure stereo correspondence and for the disparity calculation to produce consistent results.

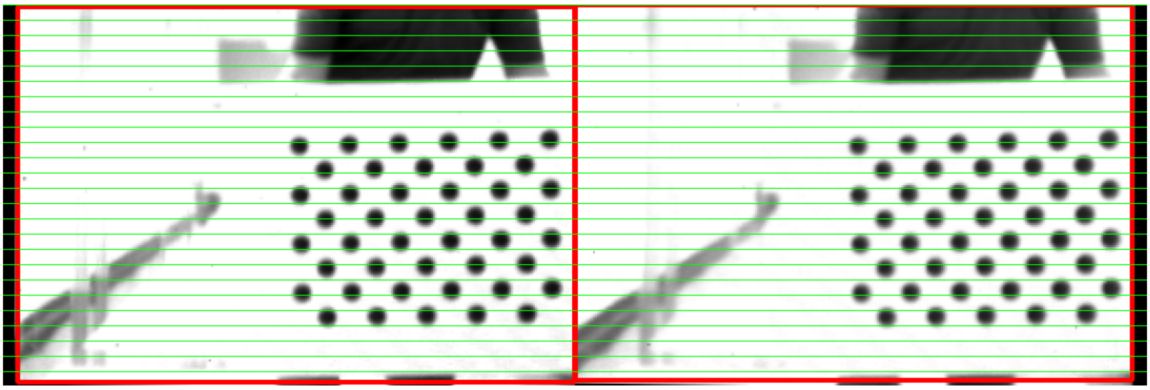


Figure 6.4: Stereo correspondence from rectified stereo images

6.3.2 Extrinsic Calibration

The extrinsic parameters consist of the rotation and translation from one virtual camera to the other. For rotation:

$$R = \begin{bmatrix} .99981 & .0004 & .01.915 \\ -.00004 & 1 & -.00001 \\ -.019 & .000004 & .99982 \end{bmatrix} \tag{6.6}$$

where R is the rotation matrix between the cameras. The translation matrix in mm between the virtual cameras is

$$T = \begin{bmatrix} -27.3978 \\ .15277 \\ -3.5641 \end{bmatrix} \tag{6.7}$$

The baseline for the cameras was determined from extrinsic calibration to be 27mm. The selection of u from 2 to 13.5 was chosen due to the loss of light near the microlens edge. A wider baseline is desired in order to maximize the ranging capability of the camera.

6.4 Multiperspective Extrinsic Calibration and Approximation

The rotation and translation of the virtual cameras is dependent on the camera parameters and the selection of the perspective rendering coordinates (u, v). This section describes an approximate equation for determining the extrinsic parameters of any (u, v) virtual camera pair given a series of stereo baseline extrinsic calibrations.

The most important parameters in this approximation are the camera vector intersection point, f_r, the translation radius, c_l, and the perspective rendering coordinates (u, v). Two assumptions are made for the extrinsic parameter approximation: the virtual cameras translate in a sphere about c_l, and the optical axis of every virtual camera intersects at a point along the plenoptic image optical axis at f_r. For cameras focused at or near infinity, a translation component is largely planar. For cameras focused in the near field, the shift of objects in the perspective images flips at the focused length of the camera, and the translation is spherical. In stereo vision, this effect results in zero disparity at the focal length. At depths in front of the focus length, the disparities are positive, while at depths behind the focus length, the disparities are negative.

In the experimental data, these results can be seen in the 135mm camera images where the camera is focused at 1.2 meters. Figure 6.5a shows the disparity at 1.5 meters. The blue indicates little to no disparity found through the stereo block matching algorithm at this depth near the focal length. Depths at further distances indicate negative disparity. Figure 6.5b shows the right image shifted laterally by 16 pixels to compensate for the negative disparity. The green in this image indicates a disparity detected through the block matching algorithm. Depths at further distances remain positive, and the block matching algorithm is able to detect distances. Another method for handling the negative disparity is through stereo rectification with the stereo calibration parameters, which will be the approach taken for this work. Figure 6.5d shows the stereo disparity for the target at 1.5 meters. The disparity map shows a similar result

(a) Disparity image with sign 1.5 meters away - no image shift

(b) Disparity image with sign 1.5 meters away - image shift

(c) Image with sign 1.5 meters away

(d) Calibrated image with sign 1.5 meters away

Figure 6.5: Target at 1.5 meters and the effect of negative disparity

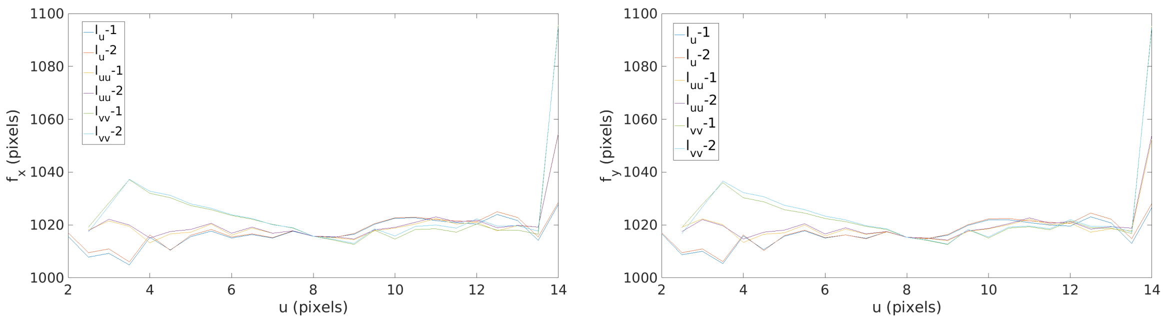
with the shifted image since stereo rectification adjusts the camera optical axes to be parallel to each other and normal to the baseline.

6.4.1 Rotation and Translation for Multiple Perspective Imagery

The approximation for the rotation and translation of a virtual stereo pair is based on an assumption that the virtual cameras translate along a sphere with a radius of the aforementioned c_l. In addition, each camera at a position along the sphere points towards a focus point at a distance f_r from the plenoptic image, or at perspective image coordinate $(u, v) = (8, 8)$. The parameters f_r and c_l can be found through knowledge of the camera or through a comprehensive extrinsic camera calibration, as described in the next section.

6.4.1.1 f_r and c_l Parameter Estimation for Stereo Rectification

Extrinsic camera calibration for a stereo pair was conducted over a series of (u, v) values as one camera and the perspective image coordinate $(8, 8)$ as the other camera. The (u, v) coordinates consisted of virtual cameras every half pixel along a $(u, 8)$ horizontal line l_h, a $(8, v)$ vertical line l_v, a (u, u) diagonal line l_{uu}, and a $(16 - v, v)$ diagonal line l_{vv} for a total of 96 calibrations. For each (u, v) camera and $(8, 8)$ camera, the extrinsic and intrinsic properties were found. Since the purpose of modeling the rotation and translation of the virtual cameras in a plenoptic camera is to remove the need for such a comprehensive stereo calibration, the intrinsic properties of each virtual camera should be the same. Figure 6.6 shows the estimated focal lengths from the calibration over the u coordinate. Most of the focal lengths keep within 10 pixels of the $(u, v) = (8, 8)$ calibrated focal length. One notable exception is at $u = 14$, where darkening of the images reduced the quality of the calibration. For the vertical image pairs, the focal lengths were similar to those shown in Figure 6.6. The other intrinsic parameter for this calibration, the principal point, was set to the image center.



(a) f_x over u coordinate

(b) f_y over u coordinate

Figure 6.6: Variation in calibrated f_x and f_y over u coordinate

Distortion over the range of the (u, v) coordinates will negatively affect the point cloud generation through a violation of the pinhole camera model. Figure 6.7 shows the k_1 distortion coefficient. Note here that pincushion distortion is positive, while barrel distortion is negative. Over most of the range of the u coordinate, the distortion is typically small pincushion distortion, but several u coordinates show barrel distortion as well. The distortion parameters, then, are difficult to model accurately over the u coordinate. For lower quality lenses such as a cheap, large plastic lens, known distortion coefficients will be much more critical in order to undistort the image and ensure a valid pinhole model. Additionally, the refocus ranging method may fail due to large distortions that result in a non constant α value at a constant depth across the (s, t) coordinates of the image. In this case, the ranging model would need an additional (u, v) coordinate input to account for that α depth change across the image. Additional experimental data would then be needed to account for variable ranging over the entire image. Alternatively, the construction of the radiance can incorporate the removal of the distortion for ensuring a plaid data structure [102]. Since the microlens images are small, the distortion can be assumed to be constant over the microlens image. The distortion correction, then, is handled through the operation to find the micro image centers and not the point coordinates [102]. For the purposes of this dissertation and due to the quality of the lens, the distortion is assumed to be small enough to be handled through the microlens calibration procedure for finding the microlens centers. Further information regarding the estimation of the distortion coefficients for a plenoptic camera can be found in a paper by Hall [124].

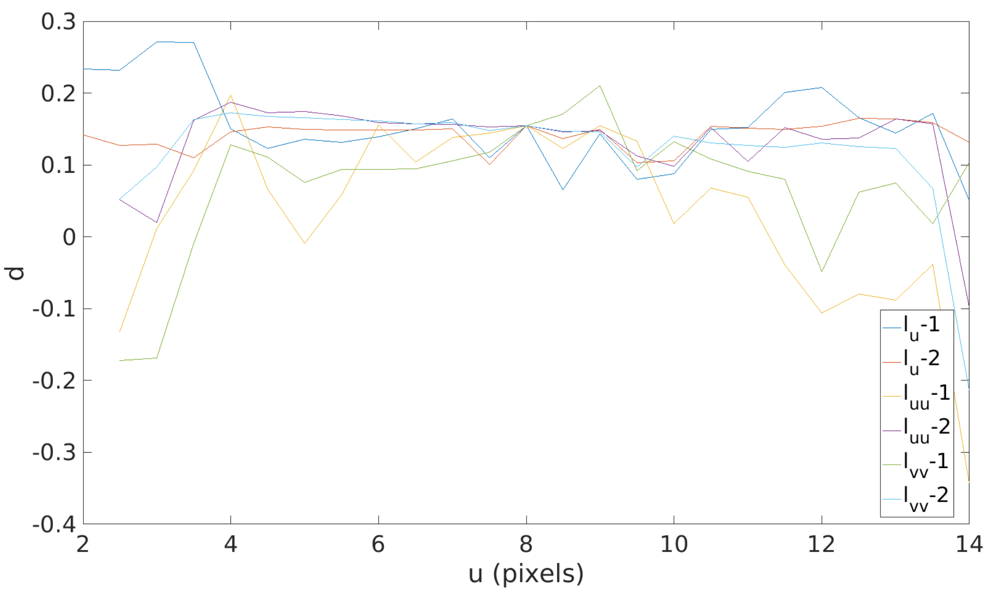


Figure 6.7: Distortion coefficient k_1 for calibration

6.4.1.1.1 Estimating c_l

The c_l parameter is the radius of a sphere about which the virtual cameras translate. Through the comprehensive extrinsic calibration, a series of translations were found for each camera. Figure 6.8 shows the translated cameras as red markings on the sphere. The particular arrangement of each of the l_u, l_v, l_{uu}, and l_{vv} lines can be seen as well. Using the coordinates of these virtual camera translations, a least squares spherical fit was used to determine the radius, c_l.

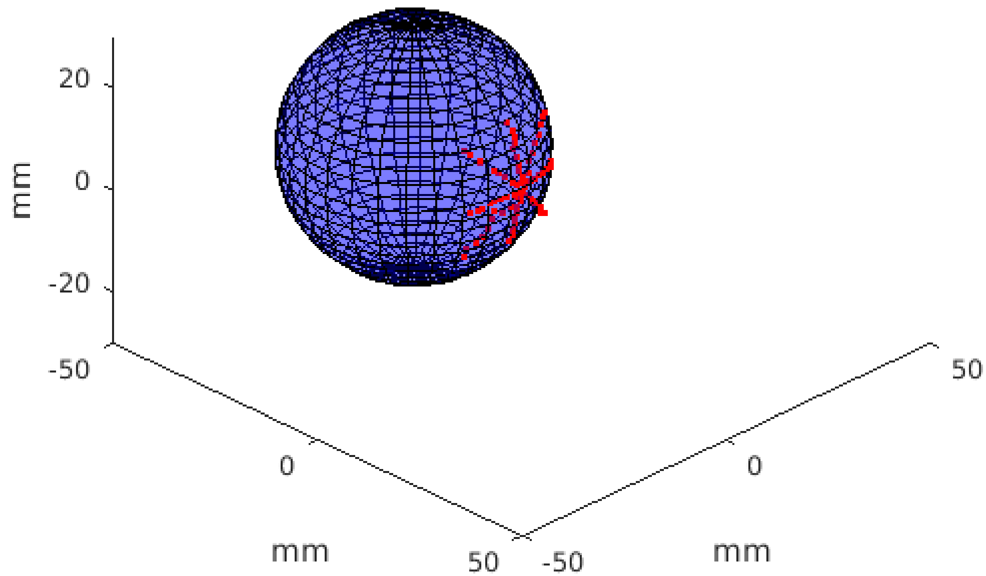


Figure 6.8: Sphere fit for determining radius

Least squares spherical fit

The spherical fit for estimating c_l followed the approach taken by Jennings [125]. The input data are n 3-D points X=[x_i y_i z_i]. The least squares fit solves $AX_c = B$ where A and B

are given in Equations (6.8) and (6.9), respectively

$$A = 2\begin{bmatrix} \sum_{i=1}^{n} \frac{x_i(x_i-\bar{x})}{n} & \sum_{i=1}^{n} \frac{x_i(y_i-\bar{y})}{n} & \sum_{i=1}^{n} \frac{x_i(z_i-\bar{z})}{n} \\ \sum_{i=1}^{n} \frac{y_i(x_i-\bar{x})}{n} & \sum_{i=1}^{n} \frac{y_i(y_i-\bar{y})}{n} & \sum_{i=1}^{n} \frac{y_i(z_i-\bar{z})}{n} \\ \sum_{i=1}^{n} \frac{z_i(x_i-\bar{x})}{n} & \sum_{i=1}^{n} \frac{z_i(y_i-\bar{y})}{n} & \sum_{i=1}^{n} \frac{z_i(z_i-\bar{z})}{n} \end{bmatrix} \tag{6.8}$$

$$B = \begin{bmatrix} \sum_{i=1}^{n} \frac{(x_i^2+y_i^2+z_i^2)(x_i-\bar{x})}{n} \\ \sum_{i=1}^{n} \frac{(x_i^2+y_i^2+z_i^2)(y_i-\bar{y})}{n} \\ \sum_{i=1}^{n} \frac{(x_i^2+y_i^2+z_i^2)(z_i-\bar{z})}{n} \end{bmatrix} \tag{6.9}$$

where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are the mean.

For the extrinsic data of the 135mm lens, the camera center, X_c, was found as shown below.

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} -2.4016 \\ 0.3146 \\ -23.9378 \end{bmatrix} \tag{6.10}$$

The radius then shown in Equation (6.11).

$$c_l = \sqrt{\frac{\sum_{i=1}^{n}((x_i - x_c)^2 + (y_i - y_c)^2 + (z_i - z_c)^2)}{n}} = 23.98 \text{ mm} \tag{6.11}$$

6.4.1.1.2 Estimating f_r

Each optical center of the virtual cameras intersect near a point in front of the camera, as seen in Figure 6.9. In this figure, the translations of the camera can be seen as the colored lines on the right side of the figure. The optical centers for each camera are the black lines that intersect at $T_z = -1500$mm. Note that the 135mm lens was set to a focus distance of 1219mm. The additional 300mm may be due to the length of the lens. The intersection point at $T_z = -1500$mm is the distance to the focus point f_r.

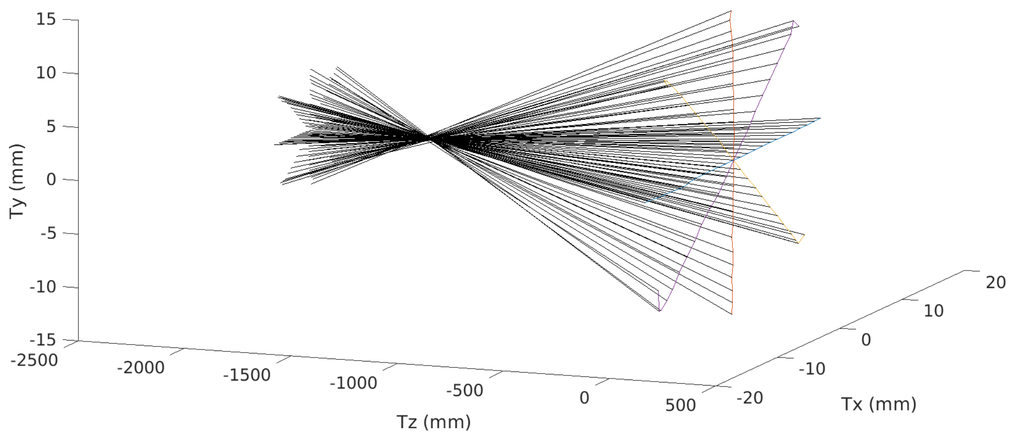


Figure 6.9: Extrinsic R and T for selected (u, v)

6.4.1.2 Calculating R and T

The rotation R and translation T are derived through the assumption of a translation on a sphere with radius c_l, where each camera is rotated towards a focus point at a distance of f_r on the optical axis. The rotation is defined in an axis-angle representation.

6.4.1.2.1 Circular segment derivation

Figure 6.10 shows the geometry of the sphere for calculating the baseline and the corresponding angle θ. The virtual cameras sit at the intersection of the sphere with the radius, r, and point towards the center of the sphere. Note here that any u or v term can be used and that the baseline itself may not be parallel to the (u, v) axis. In this case, the baseline consists of two points around a circle formed by the intersection of the sphere with radius c_l and a plane whose points are selected as the (u_1, v_1) and (u_2, v_2) coordinates. In Figure 6.10, s is the arc length, r is the radius and is either f_r or c_l, b is the baseline of the virtual stereo pair, θ is the angle that forms the rotation from the center of the sphere to the virtual cameras. The baseline b is a function of the focus length and angle θ [126].

$$b = 2f \sin \frac{\theta}{2} \tag{6.12}$$

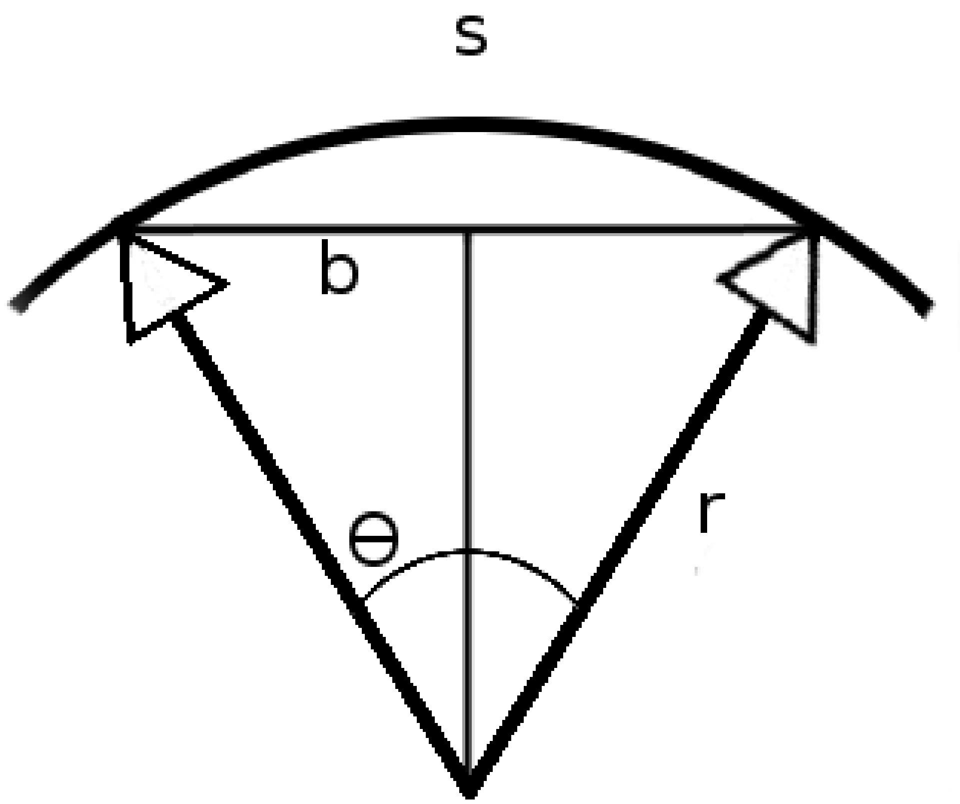


Figure 6.10: Calculating the baseline and θ

Inversely, the angle θ as a function of the baseline is seen below.

$$\theta = 2\sin^{-1}\frac{b}{2f} \tag{6.13}$$

Both Equation (6.12) and Equation (6.13) are independent of the coordinate axes of the sphere and rely only on the radius of the sphere and either the baseline or θ. The baseline is defined in (u, v) coordinates in Equation (6.14)

$$\Delta_{UV} = \sqrt{\Delta u^2 + \Delta v^2} \tag{6.14}$$

where

$$\Delta u = u_1 - u_2 \tag{6.15}$$

$$\Delta v = v_1 - v_2 \tag{6.16}$$

The angle formed by Δu and Δv is $\Delta\theta$.

$$\Delta\theta = \tan^{-1}\frac{\Delta v}{\Delta u} \tag{6.17}$$

6.4.1.2.2 Translation

The rotation and translation of the cameras consist of a rotation on a sphere whose magnitude is a function of the (u, v) coordinates. A relation between the (u, v) coordinates and the real world is needed to relate the chosen (u, v) coordinates with a real translation. The maximum θ is a function of the maximum calibrated baseline, b_{max}

$$\theta_{max} = 2\sin^{-1}\frac{b_{max}}{2c_l} \tag{6.18}$$

where b_{max} and c_l were found through extrinsic calibration or knowledge of the camera hardware. The maximum θ can be divided by the corresponding pixel width p_{max} from that maximum baseline to get the change in θ per pixel coordinate.

$$\theta_{uv} = \frac{\theta_{max}}{p_{max}} \tag{6.19}$$

Similarly, the angular offset from the coordinate axes centered on $(u, v) = (8, 8)$ is found through the offset of the (u, v) coordinate and the center coordinate.

$$\begin{aligned} u_{off} &= u_2 - 8 \\ v_{off} &= v_2 - 8 \end{aligned} \tag{6.20}$$

The offset angle from the coordinate axes is then found using θ_{uv}.

$$\begin{aligned} \theta_u &= \theta_{uv} u_{off} \\ \theta_v &= \theta_{uv} v_{off} \end{aligned} \tag{6.21}$$

A particular (u, v) uses the change in θ per pixel coordinate along with the change in pixel coordinate to calculate the baseline, b_t.

$$b_t = 2c_l \sin \frac{\Delta_{uv}\theta_{uv}}{2} \tag{6.22}$$

This baseline does not need to lie horizontally or vertically. With this new baseline, a corresponding θ_t can be found.

$$\theta_t = 2 \sin^{-1} \frac{b_t}{2c_l} \tag{6.23}$$

The translation through the coordinate axes, then, is the projection of the baseline b onto the coordinate axes.

$$\begin{aligned} T_x &= b_t \cos(\Delta\theta) \cos(\theta_v) \\ T_y &= b_t \sin(\Delta\theta) \cos(\theta_v) \\ T_z &= (c_l - c_l \cos\frac{\theta_t}{2}) \sin\theta_u \end{aligned} \tag{6.24}$$

For the 135mm lens, the maximum baseline of the plenoptic camera is $b_{max} = 27$mm. The chosen coordinates were $u = 2$ to $u = 13.5$ with no vertical component for a $p_{max} = 13.5$. Finally, the radius of the sphere as previously mentioned for the translation calculation was c_l. The vertical case where $\Delta u = 0$ is similar to Equation (6.24).

$$\begin{aligned} T_x &= 0 \\ T_y &= b_t \cos(\theta_v) \\ T_z &= (c_l - c_l \cos\frac{\theta_t}{2}) \sin\theta_u \end{aligned} \tag{6.25}$$

6.4.1.2.3 Rotation

The rotation equations follow similarly to the translation equation except that the radius is f_r rather than c_l. As such, the θ_{max} term is different.

$$\theta_{max} = 2 \sin^{-1} \frac{b_{max}}{2f_r} \tag{6.26}$$

The resulting equation for θ_{uv} was given previously in Equation (6.19). The rotation from each axis is the component of the rotation along each coordinate axis. The component of the baseline along the X axis, or u coordinate, is as provided below.

$$u_{diff} = \Delta_{uv} \cos\theta_{uv} \tag{6.27}$$

$$v_{diff} = \Delta_{uv} \sin\theta_{uv} \tag{6.28}$$

The resulting components of the baseline becomes Equation (6.29).

$$\begin{aligned} b_u &= 2f_r \sin\left(\frac{u_{diff}\theta_{uv}}{2}\right) \\ b_v &= 2f_r \sin\left(\frac{v_{diff}\theta_{uv}}{2}\right) \end{aligned} \tag{6.29}$$

The equations for rotation, then, are given in Equation (6.13) with the baseline components.

$$\begin{aligned} R_X &= 2\sin^{-1}\left(\frac{b_v}{2f_r}\right) \\ R_Y &= 2\sin^{-1}\left(\frac{b_u}{2f_r}\right) \\ R_Z &= 0 \end{aligned} \tag{6.30}$$

Notice that no rotation about the z axis is present in the rotation. The angles in Equation (6.30) are converted into a rotation matrix through the Rodrigues formula [127].

The directions of both the translation and rotation are set through the (u, v) coordinates. The rotation, R, and the translation, T, are used for stereo rectification to ensure the optical axes were parallel and normal to the baseline. The stereo rectification algorithm, implemented using OpenCV's `stereoRectify` function, transforms the images so that the matched points along the virtual camera ring resulted in either a vertical or horizontal disparity depending on whether the $\Delta\theta$ angle is closer to vertical or horizontal. With the matched features from the rectified images, a disparity was found for each pair along the virtual camera ring, and a range was calculated as described previously in Section 5.3.3.2.

6.4.1.3 Extrinsic Approximation with the 55mm Lens

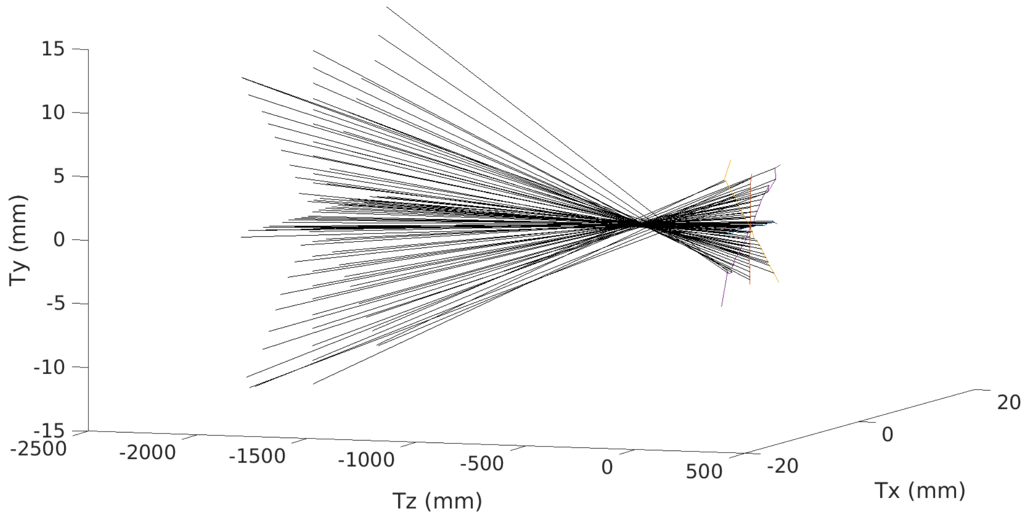


Figure 6.11: Extrinsic calibration with the 55mm lens

Along with the 135mm lens calibration given in Section 6.4.1.1, the 55mm lens was extrinsically calibrated over the vertical, horizontal, and diagonal camera sets. Figure 6.11 shows the extrinsic results for the rotation and translation of the virtual cameras of the 55mm lens. Notice that the extrinsic calibration is not as well defined as the 135mm extrinsic calibration. The rotation equations are given in Section 6.4.1.2.3 with $f_r = 500$. For this dataset, the plenoptic camera was focused at infinity, and the corresponding rotation and translation are wider such that Figure 6.11 shows the reflection of the calibration. Note that the translational virtual cameras do not exhibit the curvature of the 135mm lens. Assuming a planar translational motion results in the translational equations given in Equation (6.31).

$$
\begin{aligned}
T_x &= \frac{b_{max}}{p_c} u_{off} \\
T_y &= \frac{b_{max}}{p_c} v_{off} \\
T_z &= 0
\end{aligned}
\tag{6.31}
$$

The maximum baseline for Equation (6.31) is b_{max} = 27mm and the maximum pixel limit is p_c = 11.5 pixels.

6.5 Refocused Method Range Calibration

For refocused image ranging, an appropriate range model is needed to calculate the range to an object based on the maximum correlation from template matching. Furthermore, the method for refocusing - either through the integral method or with the FFT method, creates different correlation curves. Three lenses were tested with the plenoptic camera for determining the relationship between α and the range, Z, for both methods of refocusing. Section 5.3.4.3 described the models and how the model parameters were found, while this section describes the range selection results for these three particular lenses. The model parameters themselves are presented in tables provided in Appendix D. The data for the range model selection was collected by pointing the plenoptic camera at a target and comparing the perspective image of that target (while cutting the portions of the image that are not part of the target) with the refocused image at a particular value of α.

6.5.1 Integral Refocusing Range Model Selection

The refocusing range model selection for the integral refocusing method consists of determining the α value that is in focus at a known distance from the camera. A range of α values were chosen at $\Delta\alpha = 0.001$ from $\alpha_{min} = -2$ to $\alpha_{max} = 8$. This wide range of α values is necessary due to the wider range of α values required by the integral method. The refocused images were chosen for speed; no anti-aliasing efforts were made and bilinear interpolation was used. The perspective image used for comparison was taken at $(u, v) = (8, 8)$.

6.5.1.1 Integral Ranging model for 55mm lens

The 55mm lens is capable of distinguishing several meters of range through the correlation method. Figure 6.12 shows the correlation curves for determining range by finding the peak of the curves. The .44 meter and .61 meter curves are significantly offset from the other curves by almost $\alpha = 2.5$. Note the inflection point around $\alpha = 2$. The maximum range of the 55mm lens is clearly near $\alpha = 1.7$ once the peaks gradually become indistinguishable from each other. The maximum correlation decreases as the distances increases until the curves converge. The variation in α from the nearest range (0.44 meters) to the furthest range (3.2 meters) is more than one α, which shows the capability of the integral method to distinguish α peak detection.

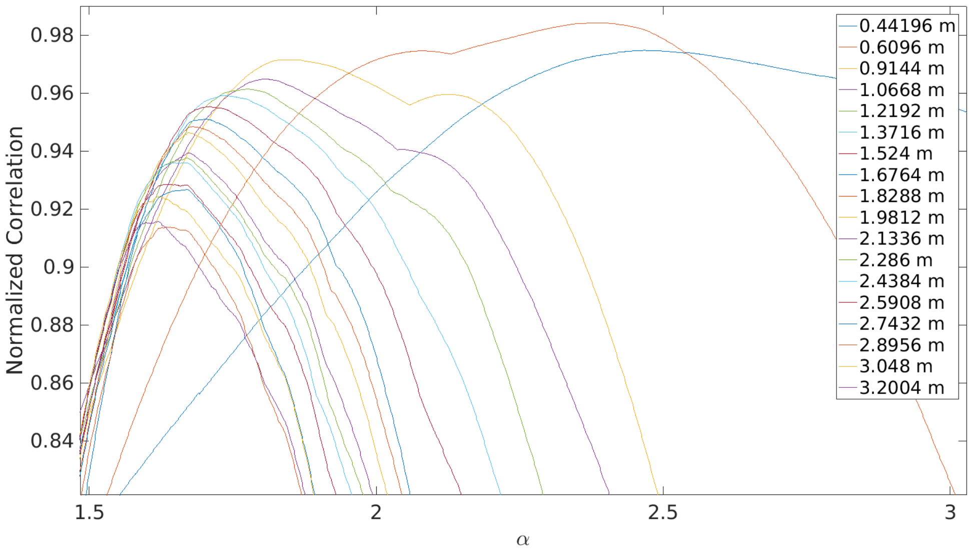


Figure 6.12: Integral method correlation curves for 55mm lens

The maximum correlation values along with the fit of each model can be seen in Figure 6.13. The maximum range at 2.5 meters of the 55mm lens using the plenoptic imagery is clear from the vertical data points around $\alpha = 1.65$. Each of the maximum correlation points approximates a smooth curve with the exception of the low α data points. The non-linear hyperbolic fit without an offset shows a significant error of 0.5 meters for the close range ($\alpha = 1.11$). The least squares exponential and least squares hyperbolic models follow the raw data with significant error of 0.5 meter error at $\alpha = 1.85$ and a poor characterization of the maximum range of the camera. The three Michaelis-Menten models also do not capture the mid-range arc. The best models through visual inspection are the nonlinear exponential model and the nonlinear hyperbolic models with an offset, which captures the maximum range as well as the mid-range around $\alpha = 1.8$.

Table 6.4 shows the integral method fit metrics for the 55mm lens. Each of the nonlinear fit models in Table 6.4 have higher R^2 values than the least squares model, and the R^2 values are approximately the same for the non-linear exponential, non-linear hyperbolic with an offset, and non-linear Michaelis-Menten models. Between the two non-linear Michaelis-Menten

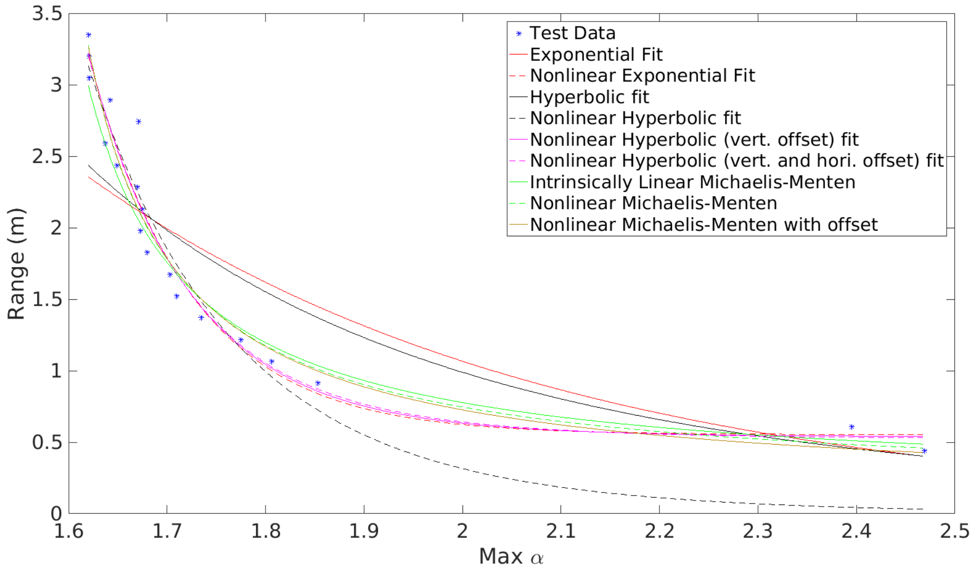


Figure 6.13: Integral method regression model curves for 55mm lens

Table 6.4: Integral Method Fit Metrics for 55mm

Goodness of Fit	**SSE** (m^2)	**SSR**(m^2)	**SST**(m^2)	R^2	**MSE**(m)	**RMSE** ($\sqrt{m}$)
$be^{c\alpha}$ (linear)	4.5984	5.959	14.1161	0.67424	0.2705	0.52009
$a + be^{c\alpha}$	0.60164	13.5145	14.1161	0.95738	0.037603	0.19391
$\frac{c}{\alpha^b}$ (linear)	3.8355	6.4821	14.1161	0.72829	0.22562	0.475
$\frac{c}{\alpha^b}$	1.1674	16.7925	14.1161	0.9173	0.06867	0.26205
$a + \frac{c}{\alpha^b}$	0.59782	13.5183	14.1161	0.95765	0.037363	0.1933
$a + \frac{c}{(\alpha-d)^b}$	0.59684	13.5193	14.1161	0.95772	0.039789	0.19947
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	1.0026	10.5051	14.1161	0.92898	0.058976	0.24285
$\frac{V_{max}\alpha}{K_m+\alpha}$	0.68394	13.2341	14.1161	0.95155	0.040232	0.20058
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	0.68126	13.4348	14.1161	0.95174	0.040074	0.20019

models, the R^2 statistical matric shows a small difference, which is expected due to the similarity of their fit curves. The mean squared error for all of the models ranges from 37 centimeters to 22.5 centimeters.

6.5.1.2 Integral Ranging model for 80mm lens

The 80mm lens has a similar distance for range discrimination using the integral method as the 55mm lens. Figure 6.14 shows the correlation curves for determining range by finding the peak of the curves. The maximum range of the 80mm lens is again seen by the convergence of the correlation curves at $\alpha = 1.25$. All further ranges converged at $\alpha = 1$, which was the max distance of the camera. The maximum α value is slightly increased to close to $\alpha = 4$.

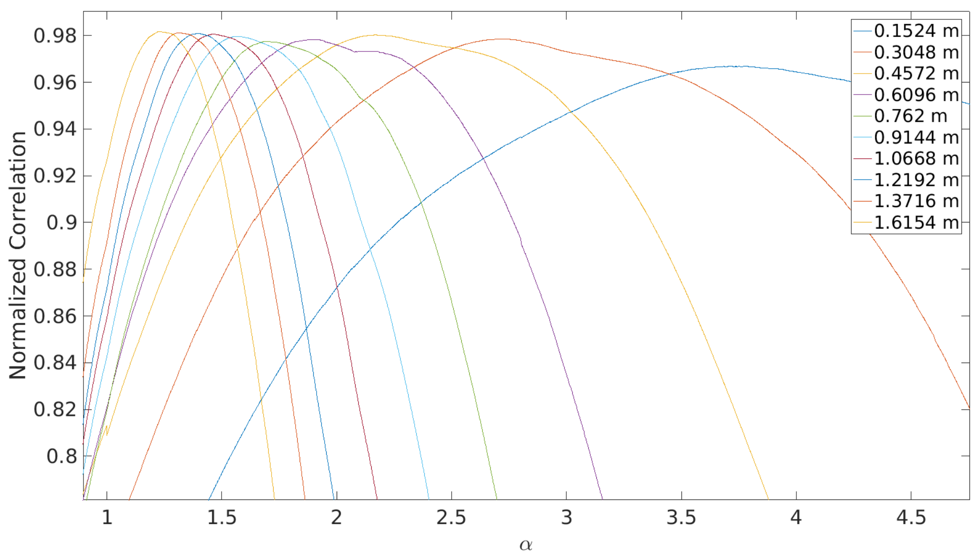


Figure 6.14: Integral method correlation curves for 80mm lens

The α values corresponding to the maximum correlation values along with the fit of each model can be seen in Figure 6.15. The maximum range of the 80mm lens using the integral refocusing method is around 2.5 meters. While the expectation of the larger lens would allow for increased range capability, the 80mm lens performs no better than the 55mm lens. However, on inspection of the correlation curves, the $\delta\alpha$ between different ranges does not show a convergence at the far range. As previously mentioned, further ranges all converge at $\alpha = 1$. This convergence around $\alpha = 1$ is detrimental to further ranging capabilities of the camera since the ability to distinguish depth is lost at $\alpha = 1$. The Michaelis-Menten models without the horizontal offset both have an erroneous asymptote. The Michaelis-Menten model with an offset follows the data with less error but has increased error in the near field. All of the nonlinear hyperbolic models follow the data well.

Table 6.5 shows the higher R^2 values for most of the models, which indicates good fits for most of the models. The intrinsically linear Michaelis-Menten model and the Michaelis-Menten model with an offset both have a negative R^2 value, which is evident by its curve in Figure 6.15. Those metrics have been removed due to the obvious poor fit. The mean squared

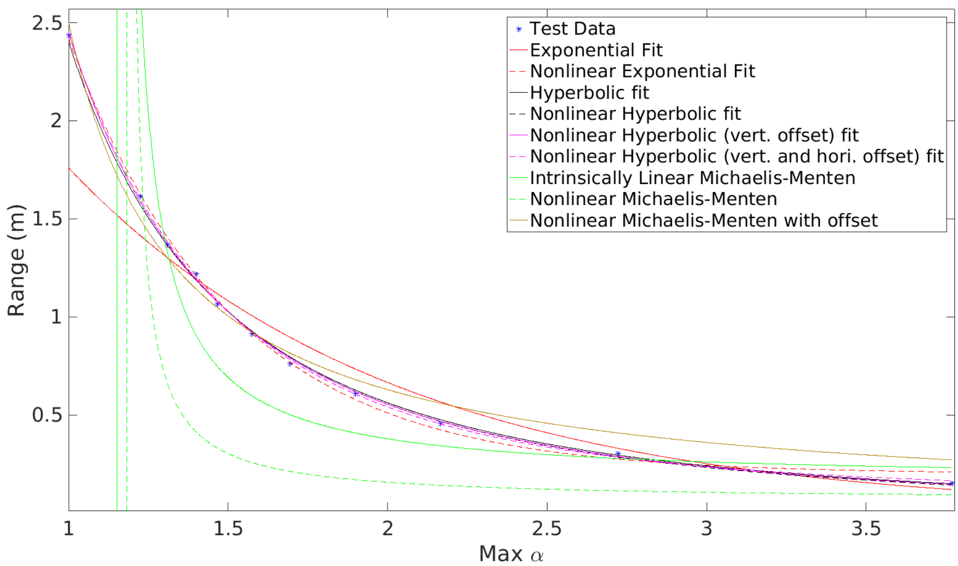


Figure 6.15: Integral method regression model curves for 80mm lens

error ranges from .4 millimeters to 6 centimeters. Except for the Michaelis-Menten models, the fit metrics for the 80mm lens is better than that for the 55mm lens.

Table 6.5: Integral Method Fit Metrics for 80mm

Goodness of Fit	**SSE** (m^2)	**SSR**(m^2)	**SST**(m^2)	R^2	**MSE**(m)	**RMSE** ($\sqrt{m}$)
$be^{c\alpha}$ (linear)	0.5645	2.3783	4.3504	0.87024	0.062722	0.25044
$a + be^{c\alpha}$	0.0085154	4.3419	4.3504	0.99804	0.0010644	0.032625
$\frac{c}{\alpha^b}$ (linear)	0.0067333	4.1457	4.3504	0.99845	0.00074815	0.027352
$\frac{c}{\alpha^b}$	0.0036441	4.356	4.3504	0.99916	0.0004049	0.020122
$a + \frac{c}{\alpha^b}$	0.0036117	4.3468	4.3504	0.99917	0.00045146	0.021248
$a + \frac{c}{(\alpha-d)^b}$	0.002309	4.3481	4.3504	0.99947	0.00032985	0.018162
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	0.07224	3.8284	4.3504	0.98339	0.0080267	0.089592

6.5.1.3 Integral Ranging model for 135mm lens

The 135mm lens is the longest lens used for this dissertation. As such, the expectation is for the 135mm lens to have the longest range capability as well. The curves in Figure 6.16 are unique among all of the lens and refocusing methods because the α term extends into the negative range. Note from Section 3.4.2.2.1 that the FFT refocusing method has no definition for $\alpha < 0$. The integral method, however, can handle negative α values as shown in Figure 6.16. These negative α values are a result of the synthetic film plane extending beyond the main lens and sensor array planes. The 135mm lens has the widest range of α values from

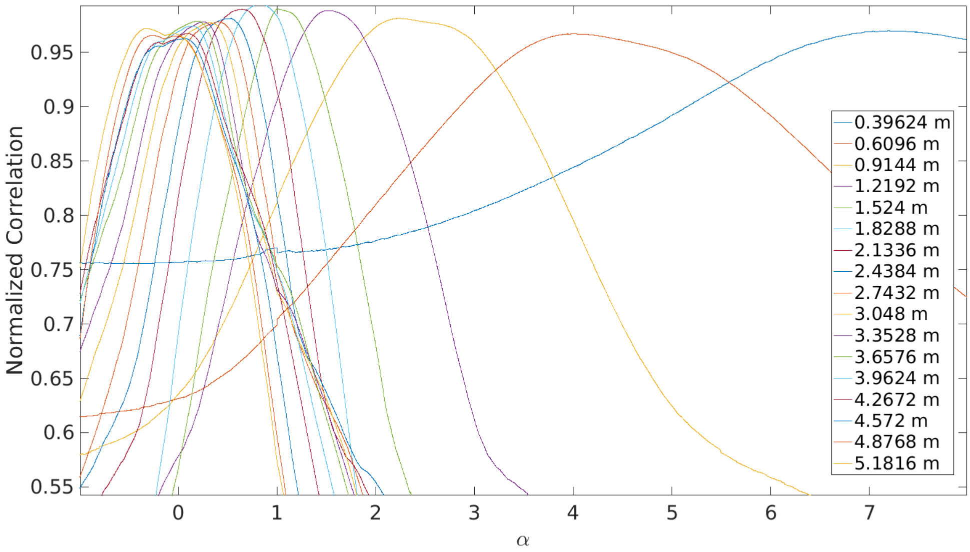


Figure 6.16: Integral method correlation curves for 135mm lens

$\alpha = -0.25$ to $\alpha = 7$. The maximum range is seen at the convergence of the correlation curves to $\alpha = -0.25$. Unlike the 55mm and 80mm lens correlation curves, the distance between the correlation peaks is distinguishable for seven of the distances. In the far range, the correlation curves have a slight double peak, which can be problematic for determining range.

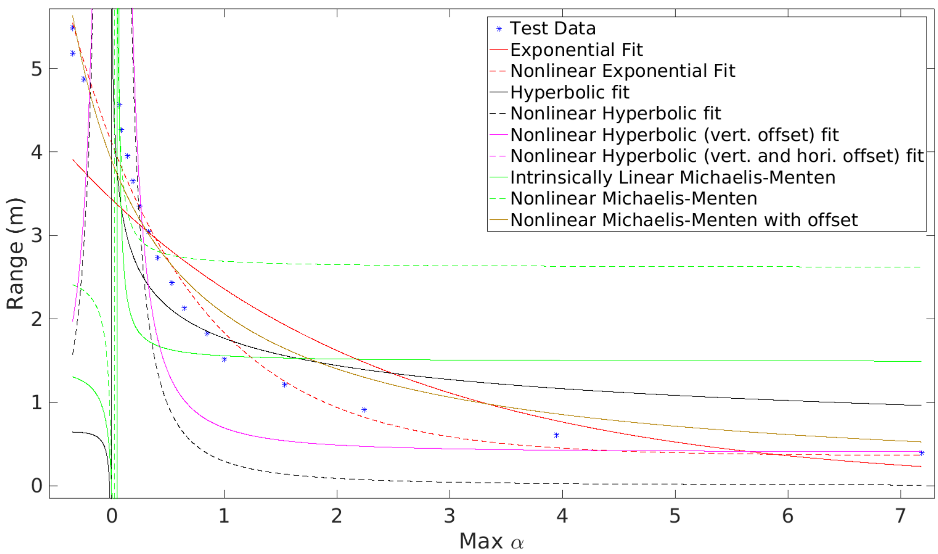


Figure 6.17: Integral method regression model curves for 135mm lens

The maximum correlation values along with the fit of each model can be seen in Figure 6.17. Most of the models failed to fit to this particular dataset due to the crossing of the vertical axis. The intrinsically linear Michaelis-Menten model, the nonlinear Michaelis Menten model, the nonlinear hyperbolic fit, and the nonlinear hyperbolic fit with a vertical offset all have their asymptote at $\alpha = 0$, and as such are inappropriate as a model for the 135mm lens. The models with horizontal offsets, the nonlinear Michaelis-Menten and the nonlinear hyperbolic model with the vertical and horizontal offsets, both succeed in capturing the curve. The hyperbolic model most closely captures the curve.

Table 6.6 shows the fit metrics for the 135mm lens for the curves shown in Figure 6.17. The maximum R^2 value for the 135mm lens is the nonlinear exponential fit with 0.97. The worst fit is the nonlinear Michaelis-Menten model and should be ignored due to its poor performance. The least squares exponential fit, like most of the least squares models, performs poorly. Finally, the Michaelis-Menton model with the horizontal offset succeeds in shifting the asympotate to handle the negative α values but cannot capture the data as well as the nonlinear exponential fit.

Table 6.6: Integral Method Fit Metrics for 135mm

Goodness of Fit	**SSE** (m^2)	**SSR**(m^2)	**SST**(m^2)	R^2	**MSE**(m)	**RMSE** ($\sqrt{m}$)
$be^{c\alpha}$ (linear)	11.0158	18.5358	44.5456	0.75271	0.68849	0.82975
$a + be^{c\alpha}$	1.3182	43.2272	44.5456	0.97041	0.087879	0.29644
$\frac{V_{max}\alpha}{K_m+\alpha}$	41.477	4.7113	44.5456	0.068887	2.5923	1.6101
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	2.8692	36.2852	44.5456	0.93559	0.17932	0.42347

6.5.2 FFT Refocusing Range Model Selection

The refocusing range model selection for the FFT refocusing method follows the same approach as with the integral method. A range of α values were chosen at $\Delta\alpha = 0.001$ from $\alpha_{min} = 0.6505$ to $\alpha_{max} = 1.8$. Compared to the integral based α range, the FFT-based refocusing method has a much narrower range of α values. The refocused images were chosen for speed; no anti-aliasing efforts were made and bilinear interpolation was used. The perspective image used for comparison was again taken at $(u, v) = (8, 8)$.

6.5.2.1 FFT Range model for 55mm lens

The 55mm lens has a short distance for range discrimination using the FFT method. Figure 6.18 shows the correlation curves for determining range by finding the peak of the curves. The .44 meter and .61 meter curves are offset from the other curves, but the maximum range of the 55mm lens is obvious once the peaks become indistinguishable from each other. The maximum correlation decreases as the distances increases until the curves converge. The variation in α from the nearest range (0.44 m) to the furthest range (3.2 m) is less than 0.1, which shows the precision required for α peak detection for the FFT based refocusing method.

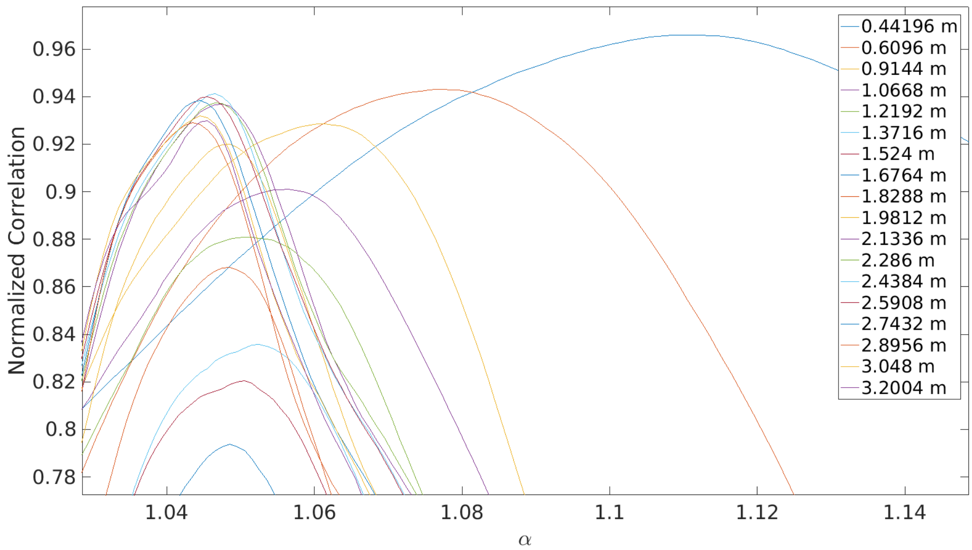


Figure 6.18: FFT method correlation curves for 55mm lens

The maximum correlation values along with the fit of each model can be seen in Figure 6.19. The maximum α range of the 55mm lens using the plenoptic imagery is clear from the vertical data points around $\alpha = 1.045$. Each of the maximum correlation points approximates a smooth curve with the exception of the data point at $\alpha = 1.05$. The nonlinear hyperbolic fit without an offset shows a significant error of 0.5 meters for the close range ($\alpha = 1.11$). The least squares exponential and least squares hyperbolic models follow the raw data with significantly error with a 0.5 meter error at $\alpha = 1.06$ and a poor characterization of the maximum range of the camera. The best model through visual inspection is the non-linear Michaelis-Menten model, which captures the maximum range and the mid-range around $\alpha = 1.06$. However, this model fails to capture the near range as well as the nonlinear hyperbolic fit with an offset and the linearized Michaelis-Menten model.

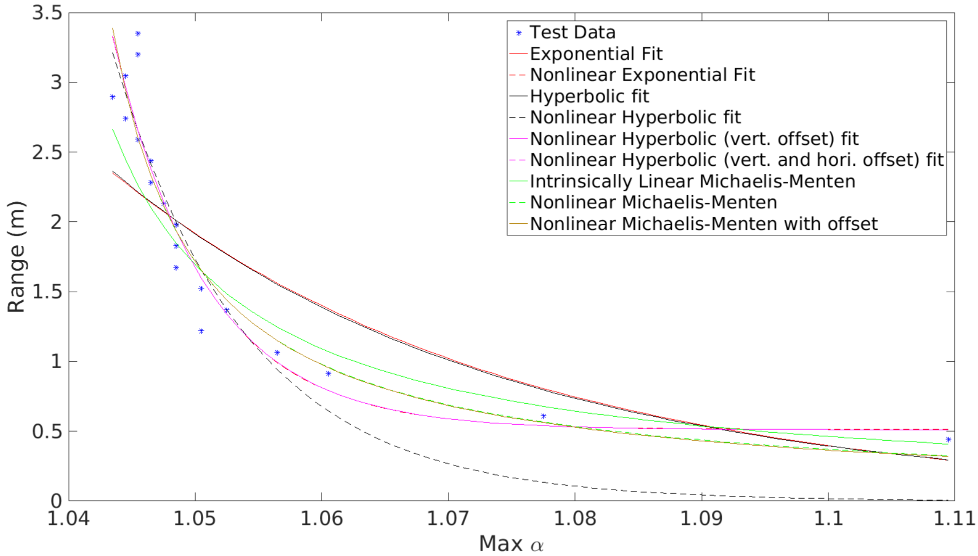


Figure 6.19: FFT method regression model curves for 55mm lens

Table 6.7 shows the nonlinear fit models where again the R^2 values are higher than the least squares model, and the R^2 values are approximately the same for the non-linear exponential, non-linear hyperbolic with an offset, and non-linear Michaelis-Menten models. Between the two Michaelis-Menten models, the R^2 statistical matric shows a significant difference, which is unexpected due to the similarity of their model parameters. However, the intrinsically linear Michaelis-Menten model does not capture the maximum range well and is most likely the cause of the lower R^2 value. The mean squared error ranges from 8 centimeters to 30 centimeters.

6.5.2.2 FFT Range model for 80mm lens

The 80mm lens has a longer distance for range discrimination using the FFT method than the 55mm lens. Figure 6.20 shows the correlation curves for determining range by finding the peak of the curves. The maximum range of the 80mm lens is again seen by the convergence of the correlation curves at $\alpha = 1.01$.

Table 6.7: FFT Method Fit Metrics for 55mm

Goodness of Fit	**SSE** (m^2)	**SSR**(m^2)	**SST**(m^2)	R^2	**MSE**(m)	**RMSE** $(\sqrt{m})$
$be^{c\alpha}$ (linear)	5.0657	5.2838	14.1161	0.64114	0.29798	0.54588
$a + be^{c\alpha}$	1.3041	12.812	14.1161	0.90761	0.081507	0.2855
$\frac{c}{\alpha^b}$ (linear)	4.9568	5.3665	14.1161	0.64885	0.29158	0.53998
$\frac{c}{\alpha^b}$	1.7983	15.7772	14.1161	0.87261	0.10578	0.32524
$a + \frac{c}{\alpha^b}$	1.305	12.8111	14.1161	0.90755	0.081561	0.28559
$a + \frac{c}{(\alpha-d)^b}$	1.305	12.8111	14.1161	0.90755	0.086999	0.29496
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	3.1754	7.11	14.1161	0.77505	0.18679	0.43219
$\frac{V_{max}\alpha}{K_m+\alpha}$	1.5398	12.2419	14.1161	0.89092	0.090575	0.30096
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	1.5381	12.2959	14.1161	0.89104	0.090478	0.3008

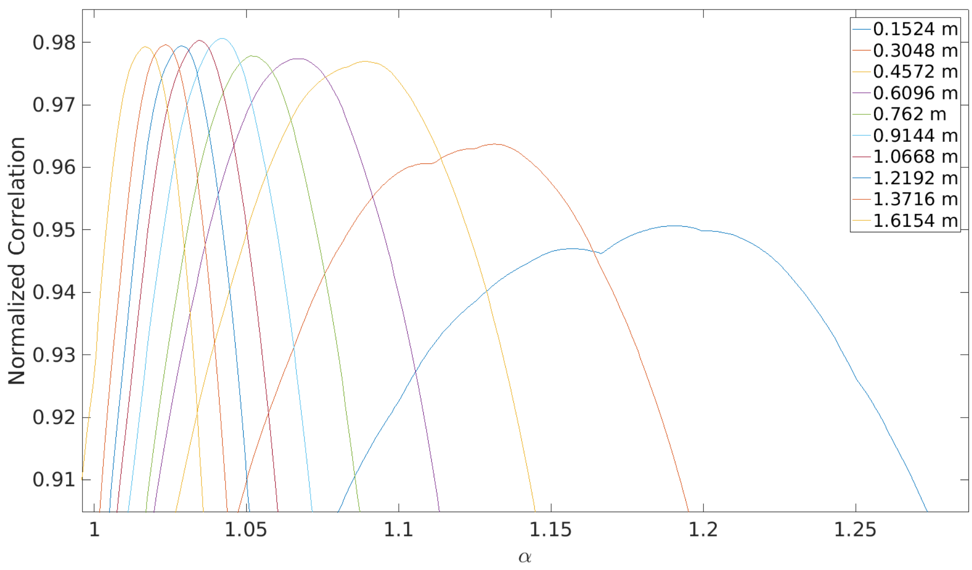


Figure 6.20: FFT Method Correlation Curves for 80mm lens

The maximum correlation values along with the fit of each model can be seen in Figure 6.21. The maximum range of the 80mm lens using the FFT-based refocusing method is around 2.5 meters. Each of the maximum correlation points shown approximates a smooth curve. Several higher range curves that are not shown become a vertical line, and those points were removed to enable a better model fit. Once again, the least squares exponential and least squares hyperbolic models do not follow the curve well with a poor characterization of the maximum range of the camera as well as two clear intersection points with the nonlinear curves at $\alpha = 1.03$ and $\alpha = 1.14$. The best model through visual inspection is the non-linear hyperbolic model with an offset, which captures the maximum range and the mid-range around $\alpha = 1.08$.

The intrinisically linearized Michaelis-Menten model fails poorly with its vertical asympotote lying between data points. Clearly, this model should not be used.

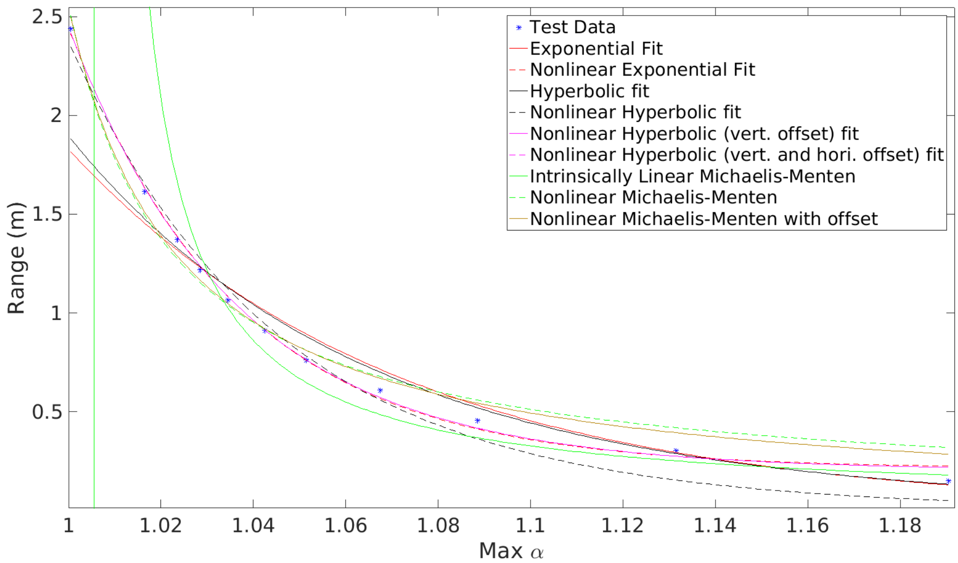


Figure 6.21: FFT method regression model curves for 80mm lens

Table 6.8: FFT Method Fit Metrics for 80mm

Goodness of Fit	**SSE** (m^2)	**SSR**(m^2)	**SST**(m^2)	R^2	**MSE**(m)	**RMSE** ($\sqrt{m}$)
$be^{c\alpha}$ (linear)	0.46537	2.6037	4.3504	0.89303	0.051707	0.22739
$a + be^{c\alpha}$	0.011974	4.3384	4.3504	0.99725	0.0014968	0.038689
$\frac{c}{\alpha^b}$ (linear)	0.3717	2.7702	4.3504	0.91456	0.0413	0.20322
$\frac{c}{\alpha^b}$	0.063663	4.7693	4.3504	0.98537	0.0070737	0.084105
$a + \frac{c}{\alpha^b}$	0.0097637	4.3406	4.3504	0.99776	0.0012205	0.034935
$a + \frac{c}{(\alpha-d)^b}$	0.0097911	4.3406	4.3504	0.99775	0.0013987	0.0374
$\frac{V_{max}\alpha}{K_m+\alpha}$	0.092034	3.7509	4.3504	0.97884	0.010226	0.10112
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	0.06362	3.851	4.3504	0.98538	0.0070688	0.084076

Table 6.8 shows each of the nonlinear fit models with higher R^2 values than the least squares model, and the R^2 values are approximately the same at 0.99 for the non-linear exponential and non-linear hyperbolic with an offset models. The 80mm dataset is a better regression fit as seen by the higher R^2 values. The intrinsically linear Michaelis-Menten model has a negative R^2 value, which is evident by its curve in Figure 6.21. The mean squared error ranges from 1.2 millimeters to 5 meters for the poor linear Michaelis-Menten fit. Except for this poor

Micaelis-Menten regression curve, the fit metrics for the 80mm lens is greater than that for the 55mm lens.

6.5.2.3 FFT Range model for 135mm lens

Like the integral version, the 135mm as the longest lens should have the longest range capability as well. Figure 6.22 shows the correlation curves for the FFT-based refocusing algorithm.

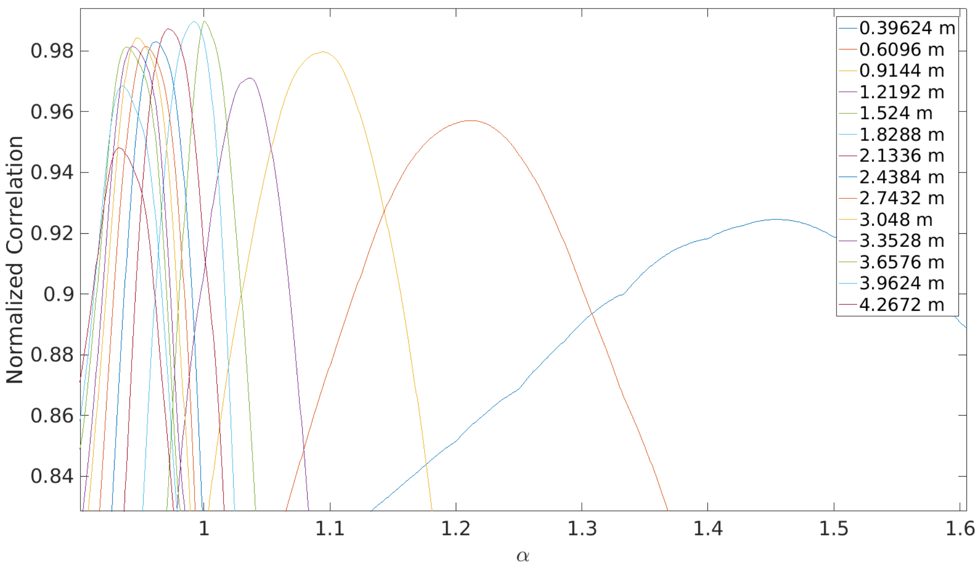


Figure 6.22: FFT method correlation curves for 135mm lens

For this lens, the correlation curves range from $\alpha = 0.95$ to $\alpha = 1.45$. The maximum range is seen at the convergence of the correlation curves to $\alpha = .95$. Unlike the 55mm and 80mm lens correlation curves, the distance between the correlation peaks is distinguishable for seven of the distances. The variation in α from the nearest range (0.39 meters) to the furthest range is about 0.5, or more than five times the variation in α for the 55mm lens.

The maximum correlation values along with the fit of each model can be seen in Figure 6.23. The maximum range of the 135mm lens using the FFT-based refocusing method is around 4 meters. Each of the maximum correlation points shown approximates a smooth curve, with the points at the furthest range having about the same maximum α. Once again, the least squares exponential and least squares hyperbolic models do not follow the curve well

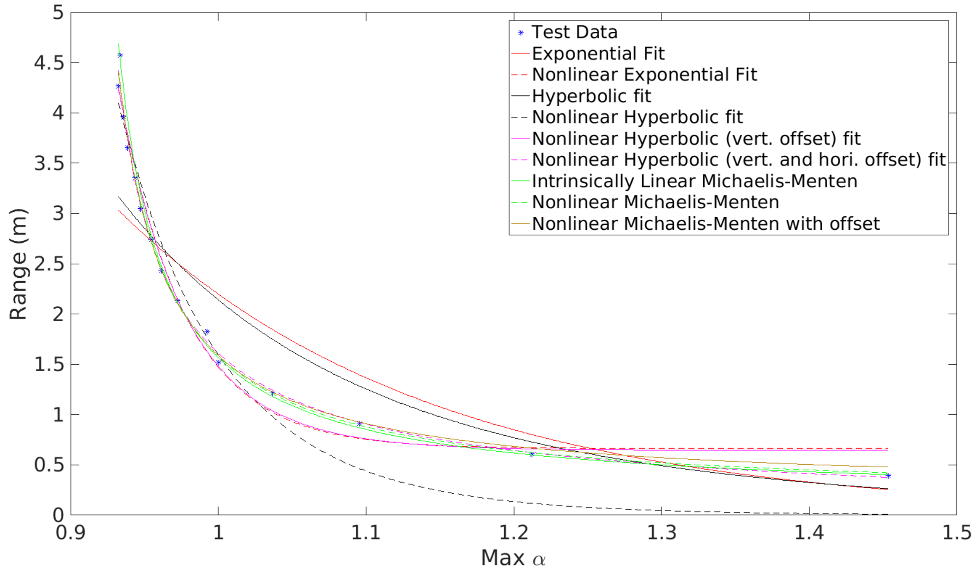


Figure 6.23: FFT method regression model curves for 135mm lens

with a poor characterization of the maximum range of the camera as well as two clear intersection points with the nonlinear curves at $\alpha = .95$ and $\alpha = 1.21$. The best models through visual inspection are the Michaelis-Menton curves, which capture the maximum range and the mid-range around $\alpha = 1.1$. Both Michaelis-Menton curves exhibit similar regression curves; the linearization for this dataset for the intrinsically linear Michaelis-Menton approach closely resembles that of the non-linear fit.

Table 6.9: FFT Method Fit Metrics for 135mm

Goodness of Fit	**SSE** (m^2)	**SSR**(m^2)	**SST**(m^2)	R^2	**MSE**(m)	**RMSE** ($\sqrt{m}$)
$be^{c\alpha}$ (linear)	7.2095	10.7015	25.6305	0.71871	0.55458	0.7447
$a + be^{c\alpha}$	0.45147	25.179	25.6305	0.98239	0.037623	0.19397
$\frac{c}{\alpha^b}$ (linear)	5.6878	12.135	25.6305	0.77809	0.43752	0.66145
$\frac{c}{\alpha^b}$	1.2374	29.9537	25.6305	0.95172	0.095188	0.30852
$a + \frac{c}{\alpha^b}$	0.41649	25.214	25.6305	0.98375	0.034707	0.1863
$a + \frac{c}{(\alpha-d)^b}$	0.14025	25.4904	25.6305	0.99453	0.01275	0.11292
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	0.41958	29.8577	25.6305	0.98363	0.032275	0.17965
$\frac{V_{max}\alpha}{K_m+\alpha}$	0.14923	25.5296	25.6305	0.99418	0.011479	0.10714
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	0.15501	25.0572	25.6305	0.99395	0.011924	0.1092

Table 6.9 shows that the maximum R^2 value for the 135mm lens is the non-linear Michaelis-Menton fit. Close behind are the non-linear exponential fit, the nonlinear-hyperbolic fit with

the offset, and the intrinsically linear Michaelis-Menten fit. The mean squared error ranges from 11 millimeters from the non-linear Michaelis-Menton curve to 437 millimeters from the hyperbolic least squares fit.

6.5.3 Summary of Range Model Selection Analysis

The range model selection and fit analysis relates the depth to the refocus variable α independent of camera parameters. With a sufficient model for range, the depth can be found at the α parameter that renders a feature in focus.

6.5.3.1 Integral-based Refocusing

The integral-based refocusing ranging results show promising ranging capabilities. The integral method shows ranging capability from 2 meters to almost 4 meters. One particular result is that the 80mm lens max range is less than the 55mm lens for ranging due to the convergence of the 80mm correlation lines at $\alpha = 1$. This feature of the correlation curves where integer values of α result in an inflection point in the curve is detrimental to the ranging fit equations. For the 55mm lens, the focus ring was set to infinity to maximize the disparity in the near field and shift the natural focal length f far from the desired depth of field. Because of the focus in the far field, the α values for the 55mm lens did not get smaller than 1.5 α at 3.2 meters and potentially avoiding the convergence problem from the 80mm lens.

Table 6.10 shows the best fit model for the integral method for relating α with distance. For the 55mm and 80mm lenses, the best fit model was the hyperbolic model with the vertical and horizontal offset. For the 135mm lens, the best model was the exponential model.

Table 6.10: Integral Method Fit Results

Lens (mm)	**Best Model**	R^2	**RMSE** ($\sqrt{m}$)
55	$a + \frac{c}{(\alpha+d)^b}$	0.95772	0.19947
80	$a + \frac{c}{(\alpha+d)^b}$	0.99947	0.018162
135	$a + be^{c\alpha}$	0.97041	0.29644

6.5.3.2 FFT-based Refocusing

Like the integral method, the 135mm lens correlation curves for the FFT method show a wider dispersion along the α axis compared with the 55mm lens. The range between the maximum and minimum α curves is much greater for the integral method than the FFT-based method. The 80mm lens range capability is between the 55mm lens and the 135mm lens range capability. However, the 80mm lens has a max range that converges at $\alpha = 1$ similar to the integral refocusing ranging analysis. The most consistent and accurate model is the hyperbolic model with a vertical and horizontal offset, which was the best model for all of the lenses for FFT-based refocusing. This hyperbolic model also adheres to the hyperbolic relationship between the disparity and the range used in stereo imaging. Table 6.11 shows the model for each lens as well as the metrics.

Table 6.11: FFT Method Fit Results

Lens (mm)	**Best Model**	R^2	**RMSE** ($\sqrt{m}$)
55	$a + \frac{c}{(\alpha+d)^b}$	0.90755	0.29496
80	$a + \frac{c}{(\alpha+d)^b}$	0.99775	0.0374
135	$a + \frac{c}{(\alpha+d)^b}$	0.99453	0.11292

6.6 Point Cloud Generation and Evaluation

Point clouds using the ranging dataset were generated using stereo ranging, multiperspective ranging, integral-based refocusing ranging, and FFT-based refocusing ranging. Features were matched from one image to another from close range to far range. For example, features from an image with the range target at 2.5 feet were matched with features from an image with the range target at 3.0 feet. This process continued until either features were no longer detected or no ranging images were available. The matched features were ranged, and a point cloud was created.

6.6.1 X-Y Axis Accuracy Evaluation

Evaluation of the accuracy of the point cloud of the X-Y camera coordinate axes uses the width of the range target in the image. While the height of the range target could also be used to evaluate the accuracy of the point cloud in the Y (vertical) direction, the boundary of the target is outside of the image in the vertical direction which limits the near range analysis. For the X (horizontal) direction, the full width (1 foot) of the sign is first seen for the 135mm camera at a distance 1.3 meters from the camera.

Fortunately, the calculation of the 3-D points for the X and Y axes is similar for each of the four ranging methods, as described in Section 5.3. For equivalent f_x and f_y focal lengths, as is the case for the experimental plenoptic camera, this evaluation assumes similar performance between the X-Y axes in generating a point cloud. Also, this evaluation will assume perfect distance to the point cloud. However, since the coordinates of the point cloud in the X and Y axes is dependent on the range, any error in the range will result in an error in the corresponding X and Y axis directions as well. This evaluation will also use hand chosen image points that lie on the edge of the sign. Since any impact by the ranging method is removed from this evaluation, this X-Y evaluation is sufficient for all of the ranging methods.

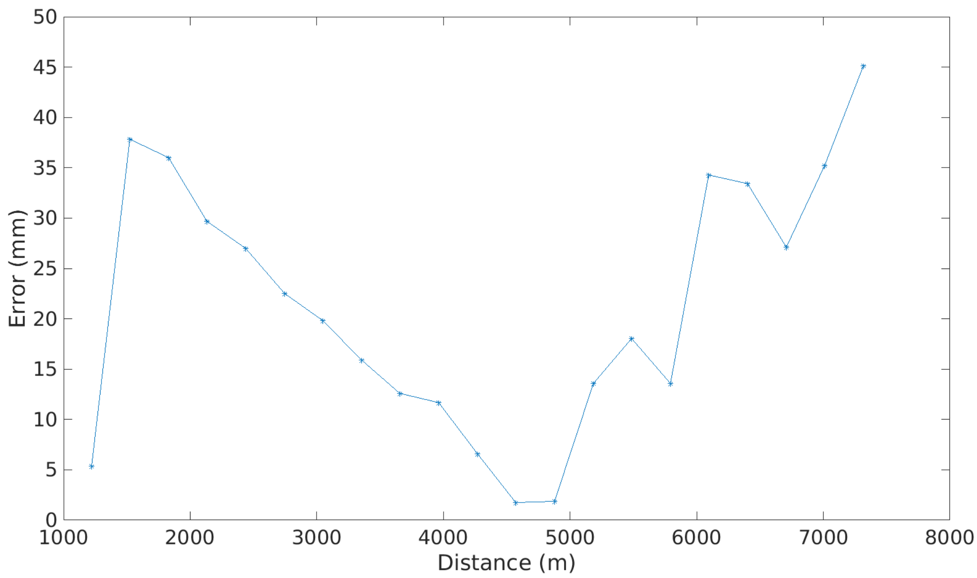
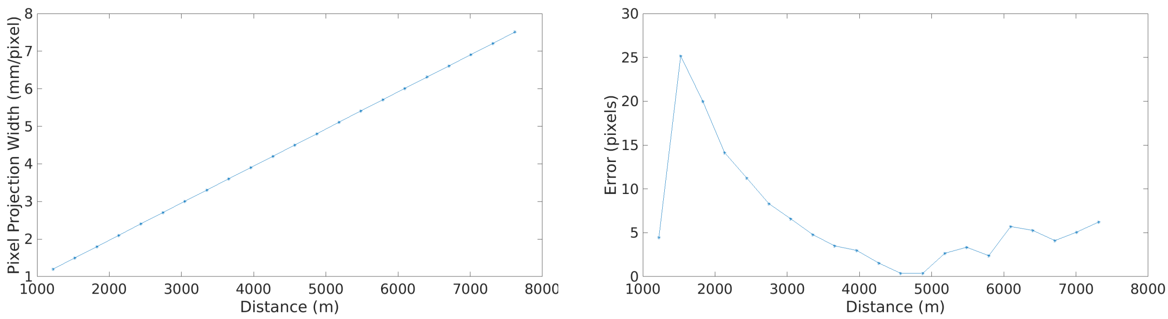


Figure 6.24: Accuracy of point cloud in the X-axis

Figure 6.24 shows the error of the estimated width of the target compared with the true width of the target at 1 foot. The error stays within 5 centimeters through a distance of seven meters. Since the range for this evaluation is perfect, the point cloud position error in the X axis does not need to increase proportionally with distance. However, each pixel represents a gradually increasing real-world length at further ranges. For the 16.05 degree horizontal field of view given in Table 6.1, each pixel represents an increasing width as seen in Figure 6.25a and Equation (6.32)

$$err_{pixels} = \frac{2Z\tan\left(\frac{HFOV}{2}\right)}{m} \tag{6.32}$$

where Z is the range, $HFOV$ is the horizontal field of view, and m is the width of the image. The error in pixels, then, can be found through division of the error in Figure 6.24 and the $mm/pixel$ found in Figure 6.25a. Figure 6.25b shows the resulting error in pixels, which represents the difference between the estimated width in pixels through reprojection of the points to 3-D and the true width of the target in pixels as reprojected into 3-D. For near ranges, the larger error results in larger pixel error most likely due to poor choice of the hand selection of the target edge, but for further ranges, the error is attributed more to the loss of horizontal lateral distance resolution from the discretization of the image.



(a) Representation of one pixel for increased distances

(b) Error of point cloud as pixel coordinates

Figure 6.25: Accuracy of point cloud in the X-axis in pixels

6.6.2 Ranging Accuracy Evaluation

For quantifying the ranging ability for each algorithm, ORB features on the range target were detected. From this point cloud, the most common range was recorded as the estimate of the distance to the target. Each of the ranging figures shows the truth in red, the estimated ranges in blue, the error in green, and finally the expected stereo ranging error bounds in dashed black. The depth error in stereo is defined as shown in Equation (6.33).

$$\epsilon_z = \frac{Z^2}{Bf}\epsilon_d \tag{6.33}$$

where Z is the depth, B is the baseline, f is the focal length in pixels, and ϵ_d is the matching error in pixels, which was set to a half pixel [128]. The depth error bounds can be viewed as a gauge of the performance of the plenoptic ranging compared with a typical stereo camera setup with the same focal length, baseline, and a matching error of 0.5 pixels. These focal lengths and baselines for both the 55mm lens and the 135mm lens were found through intrinsic and extrinsic calibrations as described in Section 6.3.

6.6.2.1 Stereo Ranging

Features were extracted from the stereo images rendered through the perspective imaging from the intrinsic and extrinsic calibration of Sections 2.3.1 and 2.3.2 and reprojected into 3-D points using the methods presented in Section 5.2. The imagery was undistorted according to the distortion coefficients found through the intrinsic calibration procedure. In the stereo block matching algorithm, the number of disparities was set to 32. Higher disparity numbers result in a larger number of depths and closer depth capabilities for wide baselines. The block size was set to 15 pixels, which was large enough to reduce random noise but low enough to retain features in the image. A bilateral filter [129] was also used to smooth the depth map but still retain edges.

6.6.2.1.1 55mm Lens

The 55mm lens failed to provide significant ranging using the stereo method. Figure 6.26 shows the stereo ranging results using ranging data with the 55mm lens. While error within the first 2 meters remains below half a meter, several distance estimates were identical at near ranges, which resulted in the flat estimate of distance. As the range exceeded 2.25 meters, the distance estimate rose significantly to almost 3.5 meters and remained there until the end of the dataset at 3 meters.

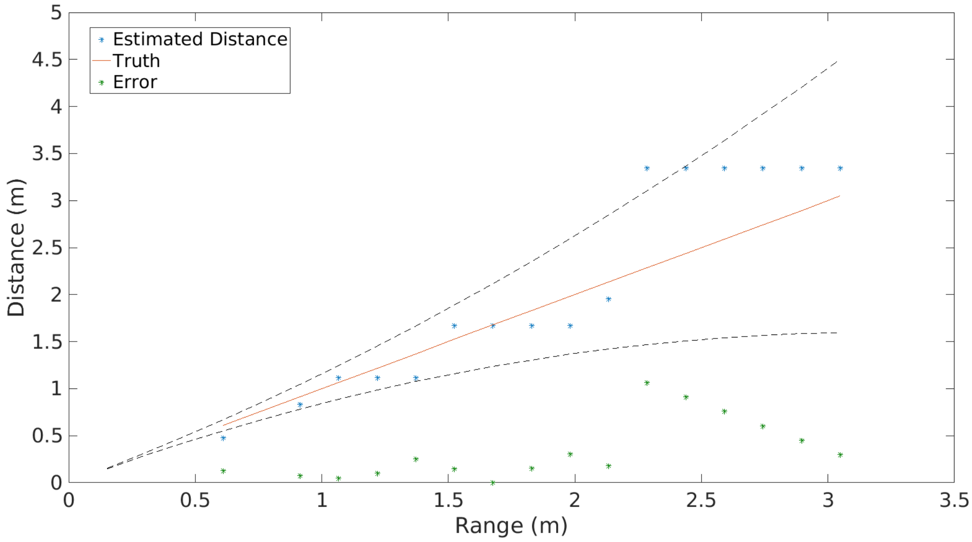


Figure 6.26: Ranging with the 55mm stereo method

6.6.2.1.2 135mm Lens

Figure 6.27 shows the reprojected point cloud for the John Deere sign at 1.5 meters with the 135mm lens. The origin at $(0, 0, 0)$ is shown as the blue star, while the reprojected points are shown as red stars. Notice that the cloud ranges to a similar Z depth. Some error not reflected in the ranging results can also be seen in the depth estimate due to scoring the most common estimate of distance as the range.

Figure 6.28 shows the experimental results from ranging using the stereo setup of the plenoptic camera for known distances with the 135mm lens. For the first few meters, the error is very small as expected for the near field of the plenoptic camera. As the range to the target exceeds 6 meters, however, the error increases past one meter. Like the 55mm lens stereo ranging, the discrete nature of far field depth is obvious further down range as different depths fall into the same estimated depth distances.

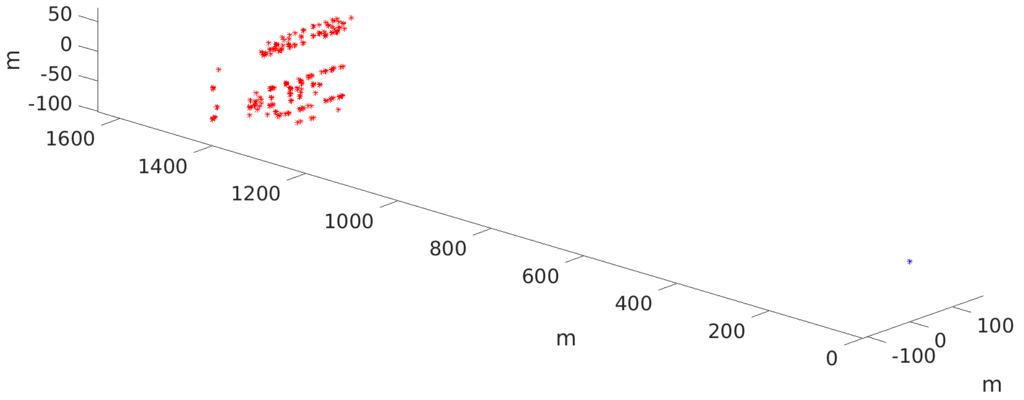


Figure 6.27: Ranging to 1.5 meters with the plenoptic camera as a stereo camera

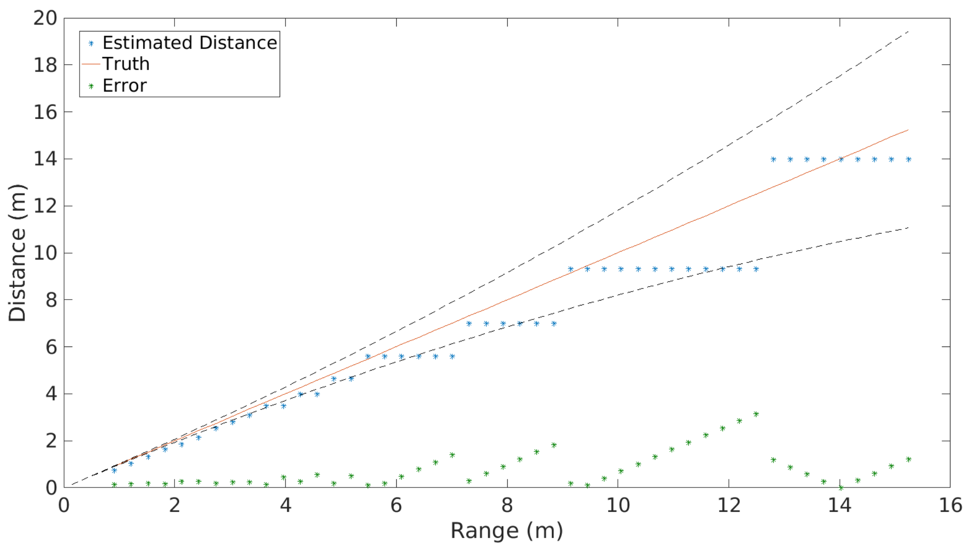


Figure 6.28: Ranging with the 135mm stereo method

6.6.2.2 Multiperspective Ranging

The multiperspective ranging method follows the methodology described in Section 5.3.3.2 with the rotation and translation of each virtual stereo pair approximated as described in Section 6.4.1. A series of 64 virtual cameras rendered images in an 11.5 pixel diameter ring. Each camera along the ring was paired with another virtual camera at uv coordinate $(8, 8)$ for each matched point. The statistical mode was used to select the best disparity among the vector of detected disparities. Note that some matched points fail to detect a disparity or even match correctly for a particular virtual camera pair. However, the 64 virtual stereo pairs ensure that enough disparities are found to remove any outliers.

6.6.2.2.1 55mm Lens

Like the stereo ranging results for the 55mm lens, the ranging performance failed to exceed 2 meters, as seen in Figure 6.29. The smaller virtual baseline between the camera pairs in the multiperspective setup is too narrow to provide enough disparity for better ranging. Past a two meter range, the estimated distance reaches its maximum capable depth at 1.5 meters.

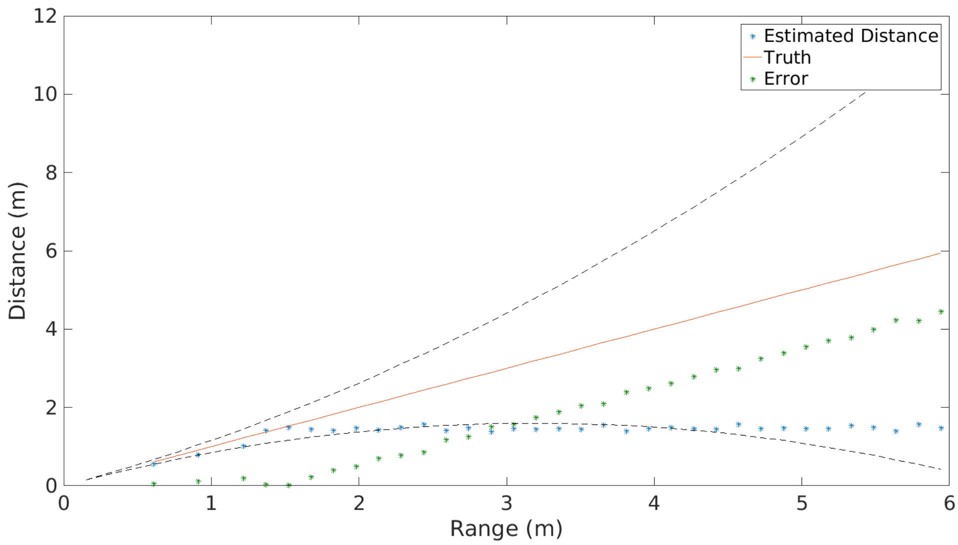


Figure 6.29: Ranging with the 55mm multiperspective method

6.6.2.2.2 135mm Lens

Figure 6.30 shows the multiperspective ranging results up to 7 meters. Since the multiperspective ranging method uses the disparity between two cameras as the core ranging equation, the ranging results should be similar to the stereo method. The additional cameras provide robustness in case image features are not matched between images. The ranging error in Figure 6.30 shows less than 1 meter error up to 6.5 meters except for an increased error around the 5.5 meter range. Figure 6.28 showed similar performance for the stereo method, as expected. While

64 cameras are sufficient for ranging, fewer cameras can provide improved computational efficiency and performance while retaining the robustness capabilities of the multiperspective ranging method.

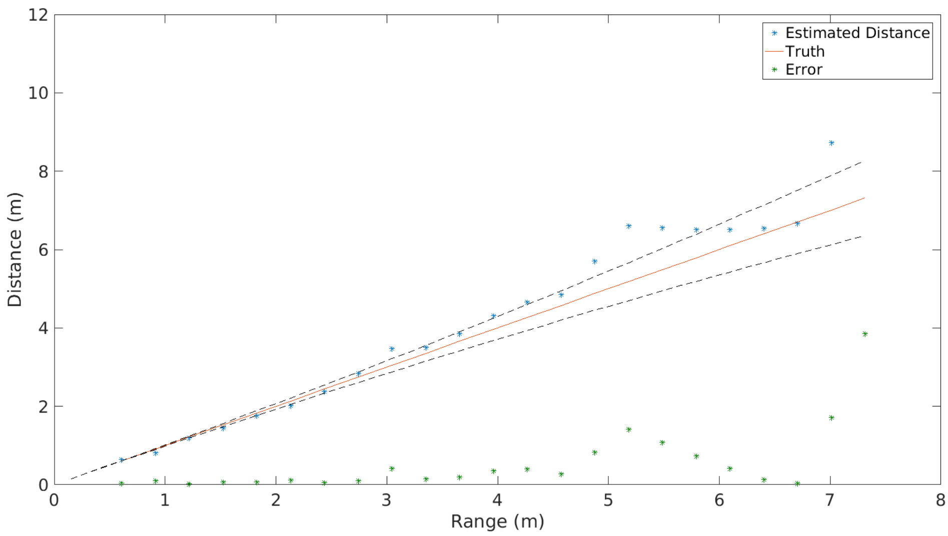


Figure 6.30: Ranging with the 135mm multiperspective method

6.6.2.3 Integral Refocus Ranging

Of the two refocusing methods, the integral refocusing method, while slower, showed a more promising ranging capability according to the results provided in Section 6.5.1. The ranging data was also used for modeling the ranging equation, but the feature point matching and 3-D point reprojection were added for these ranging results. Additionally, the templates for template matching were 15$\times$15 square sub-images rather than the entire target. The resolution of α refocusing was .1 and ranged from -2.0 to 8.

6.6.2.3.1 55mm Lens

The 55mm lens has not provided significant ranging results for any of the virtual camera methods (stereo or multiperspective). Figure 6.31 shows that the integral ranging ability of the 55mm lens is not significantly better. An inadequate ranging model for the 55mm lens, as seen in Table 6.4, results in significant noise in the estimate of the distance to the target.

However, the error remains below one meter for the maximum range for the 55mm integral ranging dataset of 3.5 meters.

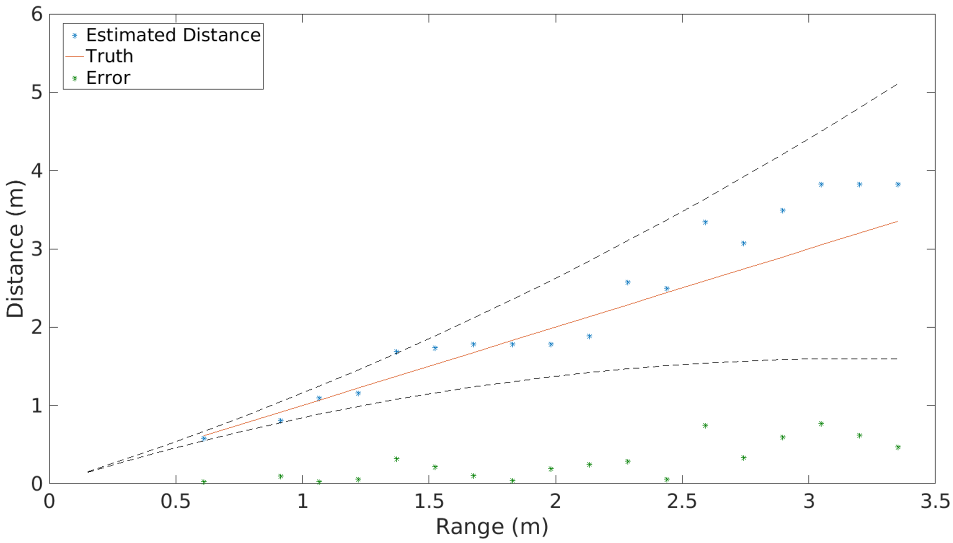


Figure 6.31: Ranging with the 55mm integral refocusing method

6.6.2.3.2 135mm Lens

Figure 6.32 shows the ranging with the integral refocusing method using the 135mm lens. The ranging error remained under 1 meter through a range of 7 meters. Most of the error in the far region consisted of ranging modeling error due to the difficulty in distinguishing the in-focus α value at those distances. Past the 5 meter range, the ORB detectors failed to match any points on the target due to the small image area of the ranging target.

6.6.2.4 FFT Refocus Ranging

The FFT refocus ranging method is significantly faster than the integral method at the cost of some loss of the 4-D light field, as described in Section 3.4.2.2.2. The ranging capability of the FFT-based method, then, is reduced as shown in Section 6.5.2. Like the integral refocusing method, the templates used for template matching were 15×15 square sub-images. The resolution of α refocusing was .001 and ranged from 0.9 to 1.45.

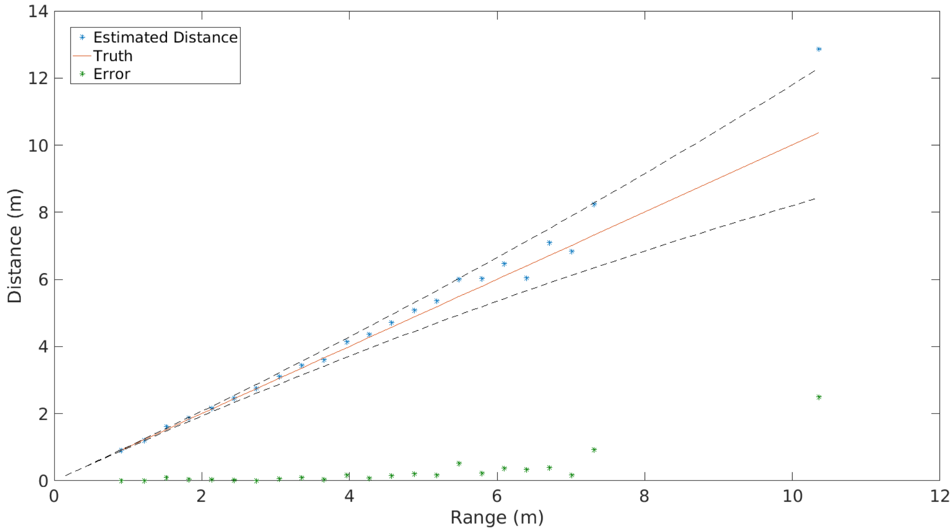


Figure 6.32: Ranging with the 135mm integral refocusing method

6.6.2.4.1 55mm Lens

The 55mm lens ranging algorithm using the FFT refocusing method resulted in less than 0.5 meter distance error through 3.5 meter range, as seen in Figure 6.33. Unlike most of the other ranging results, the distance estimate error did not gradually increase as the range increased. Around 1 meter, the error spikes up to 0.5 meters, then stays less than 0.1 meters until a range of 2 meters.

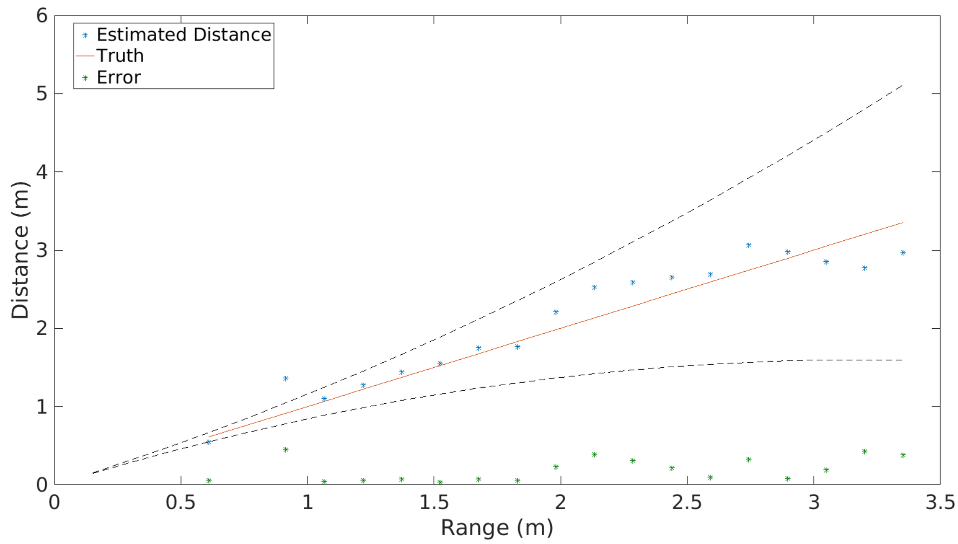


Figure 6.33: Ranging with the 55mm FFT-based refocusing method

6.6.2.4.2 135mm Lens

Like the 135mm integral refocusing method, the performance of the 135mm FFT refocusing method, shown in Figure 6.34, was highly dependent on the quality of the ranging model described in Section 6.5.2.3. The ranging error past 4 meters showed significantly small ranging error of less than 0.1 meters. The error then increased to about 2 meters at around the 6 meter range. At these ranges, the ranging model cannot adequately model the estimate distances for a given α.

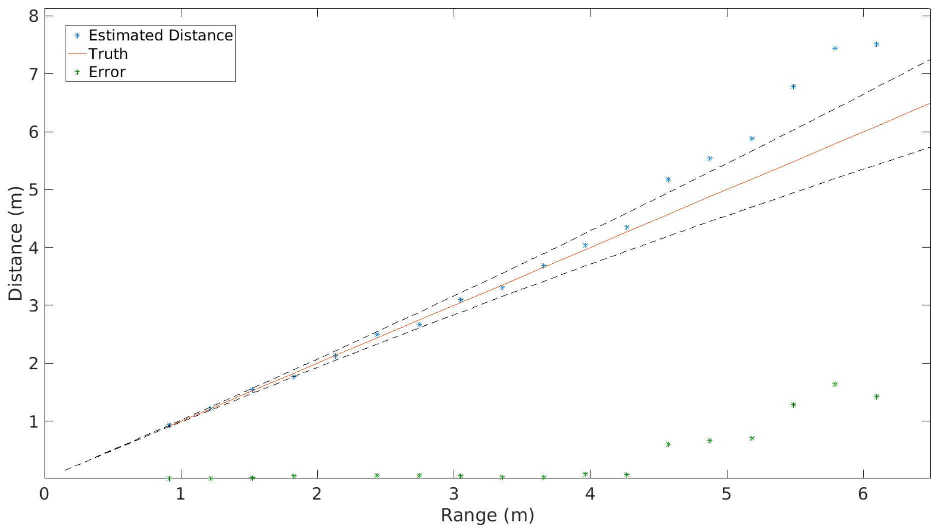


Figure 6.34: Ranging with the 135mm FFT-based refocusing method

6.6.2.5 Accuracy Ranging Summary

Most of the information available in the plenoptic images imaged through the 55mm lens provides a wider field of view than the 135mm lens, while the 135mm lens renders improved depths and greater perspective differences.

6.6.2.5.1 55mm Lens

The 55mm lens on the plenoptic camera did not provide the long ranging capabilities required for navigation. While depth discrimination is viable within 2 to 3 meters, a measurable range using all four ranging methods could only provide about a half meter accuracy. Every feature in a vision navigation system using the 55mm lens would need to be within these 2 to 3 meters. For the stereo case, depth error exceeded 0.5 meter, which results in indiscriminate ranging for multiple range targets, past only 1 meter. The perspective ranging also showed indiscriminate ranging capability at a range of 1.5 meters. Both the integral and FFT-based ranging gave improved ability to range but only to a distance of about 3 meters.

6.6.2.5.2 135mm Lens

The 135mm lens provided a significantly greater ranging capability. The stereo method provided less than a half meter error at a range of 6 meters. At ranges past 6 meters, the ranges failed to discriminate for each range as the ranging error exceeded half of a meter. The multiperspective method estimated ranges out to 6.5 meters with the exception of a large error around 5 meters. The integral ranging method provided less than half of a meter resolution past 6 meters. The FFT refocusing method provided less than a half meter error up to 4 meters.

6.6.2.5.3 Stereo Ranging Error

The stereo ranging bounds are shown as the black dashes in each of the 55mm and 135mm lens ranging results. The much wider ranging error bounds of the 55mm lens show a reflection of the narrower baseline and smaller focal length of the 55mm lens compared with the 135mm lens. As expected and seen in the ranging results, the expected ranging error for the 55mm lens is larger than the 135mm lens. One example of the wide ranging error can be seen in Figure 6.29, where the range estimate is no longer able to distinguish between different ranges due to the similarity between the two images and becomes constant. The stereo ranging error bounds, however, follow closely along with the estimated distance even when it is constant. In Figures 6.28 and 6.26, the stereo ranging results for the 55mm and 135mm lens show the expected stairstep pattern from the discrete pixel disparity. The ranging error bounds in the 135mm stereo ranging figure almost fully capture the stairstep pattern. With a matching disparity of 1.0 pixels instead of 0.5 pixels, the stairstep pattern is completely enclosed. Similarly, the FFT refocused ranging results with the 135mm lens captures additional data points inside of the stereo ranging error bounds when the matching disparity is 1.0. Beyond those points, modeling error in the ranging model is dominant as the data in the vertical asymptote of the ranging model is not captured well.

6.6.3 Overview of Ranging with Rendered Images

Four ranging methods have been described in this chapter: stereo ranging with perspective images, multiperspective ranging with perspective images, integral refocusing ranging with the integral refocused images and perspective images, and finally FFT refocusing with the FFT-based refocused images and perspective images. These ranging methods each have advantages and disadvantages depending on the particular scenario for ensuring low rendering times and desired performance.

6.6.3.1 Selection of Ranging Approach

As described in Chapter 4, perspective image rendering is fast with computation times in the millisecond range, but both refocused image rendering methods are slower in the tens of milliseconds. For this discussion, computation times will be discussed according to the GPU computation times, since the serial and OpenMP refocused image rendering computation times are not real-time capable. Note here that for the particular plenoptic camera used in this dissertation, real-time capable is defined to be 4-5 Hz, which was the maximum frames per second for recording imagery with the camera. Of course, bandwidth improvements have increased the maximum image rate significantly [26].

6.6.3.1.1 Stereo Ranging

The stereo ranging algorithm requires two perspective images. Using the perspective from plenoptic approach, the serial, OpenMP, and CUDA cases are all able to provide two perspectives efficiently. For systems without a GPU or more than one core on a processor, the stereo ranging algorithm is a feasible, real-time ranging solution with the plenoptic camera. Unfortunately, the use of only two perspectives severely limits the perspective capabilities of the plenoptic camera, and most of the information within the plenoptic image is not used. As such, the stereo ranging approach should only be used when processing power is limited.

6.6.3.1.2 Multiperspective Ranging

The multiperspective ranging approach takes advantage of the multiple perspective capabilities of the plenoptic camera while maintaining the efficiency of the perspective rendering methods. One design consideration is the number of perspectives that are ultimately desired for ranging. In the case of many perspectives, the perspective rendering from the plenoptic image may not be as efficient as the perspective rendering from the radiance since successive perspectives can be generated from the radiance more efficiently than the raw plenoptic image. The number of desired perspectives, however, must be large enough for the time to generate the radiance to be smaller than the longer computation time of rendering the perspective images directly from the plenoptic image.

The multiperspective ranging approach is still valid for systems without a GPU or multiple processors. While adding additional perspectives requires feature detection and matching, the small rendering times for more perspective images means the multiperspective ranging approach is an efficient way to more robustly determine range using multiple perspectives. Ultimately, the processing power available to the algorithm designer will dictate how many perspective images to use in the multiperspective ranging algorithm.

6.6.3.1.3 Integral Refocus Ranging

Of all of the ranging algorithms, integral refocused ranging had the largest range at around 6.5 meters. Most of this additional range came from the ability of the ranging algorithm to accurately discriminate different in-focus α depths. Unfortunately, the integral refocus ranging approach is also the slowest approach for creating a focal stack of various α refocused images with a computation time of 11.3 ms for each α refocused image after radiance generation. In addition, the integral refocus ranging approach is only viable on a GPU-based architecture since the computation time on OpenMP for rendering a single integral image is 217 ms. Of the two refocus ranging approaches (the α-sweep method and the autofocus method) the α-sweep method is best for ensuring real-time operation because the algorithm designer is able to choose the resolution and number of refocused images in the focal stack. Like the multiperspective approach, the processing power available to the hardware designer will dictate the α (or depth) resolution of the integral refocused ranging approach.

6.6.3.1.4 FFT Refocus Ranging

While the integral refocus ranging approach is accurate, the FFT-based refocus ranging approach is fast in calculating a focal stack of α refocused images. After the initial overhead of calculating the shifted radiance and 4-D FFT, the successive slices and inverse FFT's are very fast with less than a millisecond required for each refocused image. As such, the α-sweep method using the FFT refocused ranging will provide greater depth resolution than the integral refocused approach. In addition, the autofocus algorithm can be used for low feature point images, where the uncertainty on the number of iterations for the golden search algorithm is not as impactful on the total processing time. Like the integral refocused ranging approach, however, the only viable real-time architecture is on a GPU due to the OpenMP processing time of 242 ms.

6.6.3.2 Efficiency of Ranging with Rendered Images

Accurate ranging is an extremely important part of vision navigation. Stereo camera ranging has been thoroughly researched, and open source software solutions are available both with multithreaded solutions and GPU solutions [20]. As such, this dissertation will not thoroughly explore the computational times of widely available software. However, the discussion of the total time required for generating a point cloud is important for ensuring that the efficient rendered imaging algorithms have short enough computation time to remain real-time capable with the addition of the ranging algorithms.

One of the biggest disadvantages of the rendered images for vision navigation is the low resolution, but the low resolution is also one of its biggest advantages due to the lower processing times. The resolution of the rendered images of 286 $\times$ 190 means significantly shorter processing times over more standard but larger image resolutions. For example, the stereo and perspective ranging algorithms require the rectification of images. This rectification process is only a 0.2 ms time commitment to warp the image to account for the rotation and translation of the extrinsic calibration. Similarly, the disparity calculation for the depth map in the stereo case as well as the bilateral filter calculation for smoothing that depth map are both only 2 ms each. Finally, reprojection of the features into 3-D requires less than 1 millisecond.

One important step for creating a sparse point cloud is finding matched feature points. Each algorithm uses matched feature points for tracking features from one timestamp to the next. The multiperspective ranging algorithm also uses matched features for calculating the disparity to each of the virtual cameras around the ring. A sample computation time for extracting and matching features is 10 ms. For the multiperspective case, this computation time is a significant burden for ensuring correct disparities among all of the virtual cameras.

It is important to note that these computation times are dependent on the scene. For example, a more feature rich environment will increase the computation time for finding the disparity in the multiperspective ranging algorithm. Nevertheless, the time for processing the stereo and refocused images is not a significant burden on calculating a sparse point cloud for the 190 $\times$ 286 resolution rendered image of the experimental plenoptic camera. For plenoptic imagery with greater rendered image resolutions, this processing time will increase significantly. As described in Section 6.6.3.1, the algorithm designer can choose the resolution of the focal stack or the number of virtual cameras to ensure real-time capability with the available computational power with a given plenoptic camera.

6.7 Target Tracking using Visual Odometry Methods

The previous section provided a ranging evaluation using tracked features from one frame to another frame and their point clouds. In this section, those tracked features and point clouds will be used to track a moving target using visual odometry methods. Note that in vision navigation, ego-motion is typically used; however, to avoid the challenges presented in Section 6.2, the rotation and translation of the ranging target as it moves downrange will be tracked over several images. The visual odometry algorithm used for tracking is the perspective n-point algorithm using RANSAC as implemented in the `solvePnPRansac` of OpenCV. The image points from one frame will be compared with the corresponding object points of the last frame, and the resulting rotation and translation is defined as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & R_{12} & R_{13} & T_1 \\ r_{21} & R_{22} & R_{23} & T_2 \\ r_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{6.34}$$

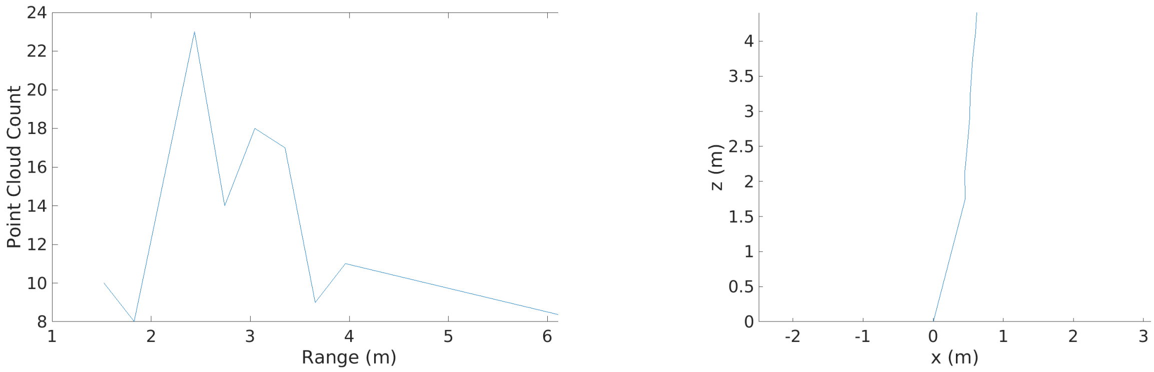
where (u, v) are image coordinates, f_x, f_y, c_x, and c_y are intrinsic matrix variables, X, Y, Z are world coordinates, R is the rotation matrix, and T is the translation vector. Note here that in the usual ego-motion visual odometry case, the inverse must be taken to track the ego motion rather than the cloud coordinates; for tracking the ranging target, the rotation matrix and translation vector can be used directly for determining position as follows in Equation (6.35).

$$\begin{bmatrix} X_t \\ Y_t \\ Z_t \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_{t-1} \\ Y_{t-1} \\ Z_{t-1} \end{bmatrix} + \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{6.35}$$

The ranging data presented previously used true ranges as a method for truth. In this visual odometry case, the origin begins at the first image frame used for estimating the rotation and translation. Since feature point matching and tracking is typically not successful in the ranging dataset until almost the full sign is visible at a few meters, the position of the camera is not the origin. Instead, this first image frame used for estimating the rotation and translation is the origin of the visual odometry path for the target. As shown previously, each of the ranging methods will be used as part of the visual odometry solution. Note that since each ranging method had varying distance capabilities and feature matching results (both in terms of first detected location and different rendered image types), a direct comparison between the different methods is not explored. In addition, only the 135mm lens is shown here; ranging results for the 55mm lens at only a few meters was not sufficient for trying to estimate the rotation and translation of the ranging target.

6.7.1 Stereo Odometry with Perspective Images

The stereo ranging algorithm presented in Section 5.2 was used as the point cloud generator for the perspective n-point algorithm. For this particular case, the image size was tripled to explore the possibility of additional points from the small 286×190 images. Figure 6.35a shows the sizes of the point cloud for each image in the dataset. The maximum point cloud size in this dataset was 23 points, which is very small for a vision navigation algorithm. This small size is a consequence of the small resolution of the rendered images. However, increasing the rendered image resolution through interpolated resizing did not result in significantly more points for the point cloud. A minimum point cloud size of 8 points was set to avoid wildly erroneous rotation and translations due to local minima. Figure 6.35b shows the position path of the ranging target from its initial position at 1.5 meters in front of the camera (as seen in Figure 6.35a). Note here that the primary motion of the target is in the z (forward) axis, as expected.



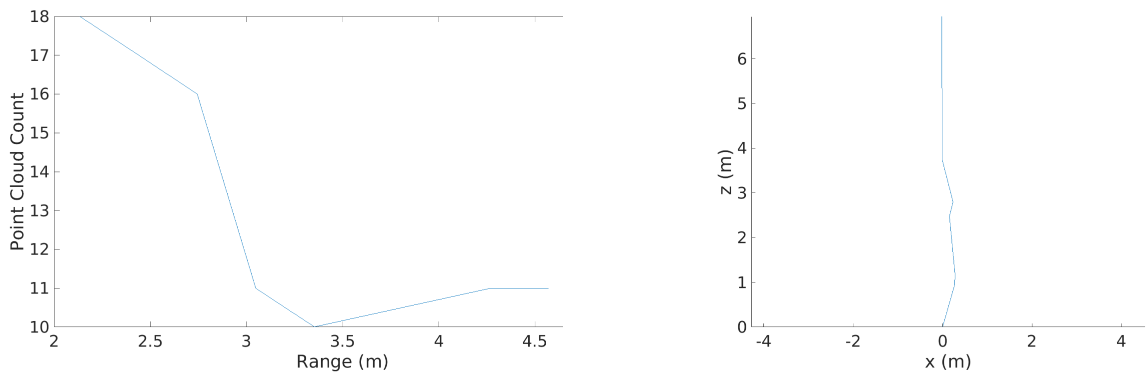
(a) Number of points in point cloud

(b) Target tracking VO path

Figure 6.35: Stereo plenoptic odometry

6.7.2 Multiperspective Plenoptic Odometry

Like the stereo algorithm, the multiperspective plenoptic odometry method uses the multiperspective ranging algorithm presented in Section 5.3.3.2 for generating the point clouds and the perspective n-point algorithm for estimating the rotation and translation of the ranging target. Unlike the stereo algorithm, the multiperspective plenoptic odometry method uses the original rendered image size of 286 × 190. A minimum point cloud size of 10 points was used for this ranging method, as seen in Figure 6.36a. Any image frame that resulted in a point cloud with fewer points would be skipped, and the rotation and translation estimated would then be from the frame prior to the skipped frame to the image frame after the skipped frame. Note that the minimum point cloud size is larger than the stereo case since the redundancy of the multiperspective algorithm allowed for handling missed matched features. Figure 6.36b shows the path of the tracked target through the ranging dataset. Like the stereo plenoptic tracking, the general motion of the tracked target was forward, as expected.



(a) Number of points in point cloud

(b) Target tracking VO path

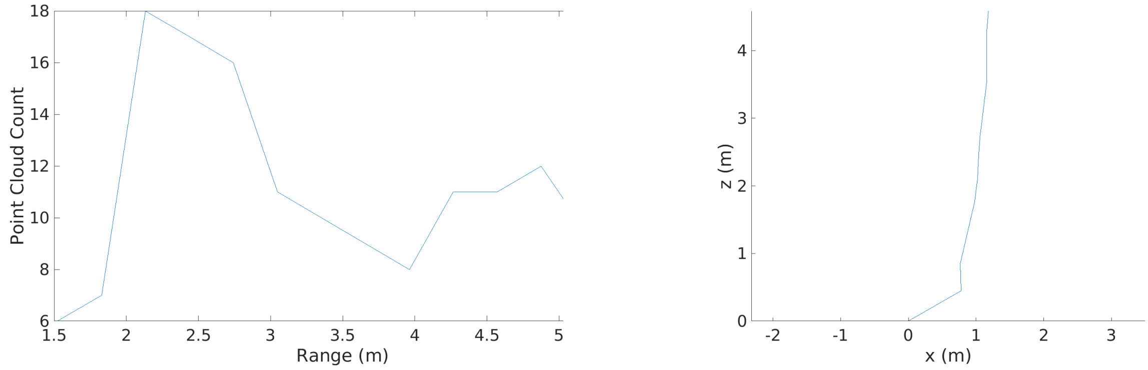
Figure 6.36: Multiperspective plenoptic odometry

6.7.3 Integral Refocusing Plenoptic Odometry

The integral ranging method, as described in Section 6.6.2.3, was the most accurate of the four ranging methods. Because of this accuracy, a minimum point cloud size of 6 was chosen under the assumption that the increased accuracy would reflect more accurate results with fewer points. Using the standard rendered image resolution size, the target path generated from the integral refocusing method is given in Figure 6.37b. After a poor positioning result at the start, the path of the targeted sign propagated in the positive z direction.

6.7.4 FFT Refocusing Plenoptic Odometry

Finally, the path for the FFT refocused plenoptic odometry, as described in Section 5.3.4.2, is given in Figure 6.38b and is similar to the path results of the other methods. The point cloud

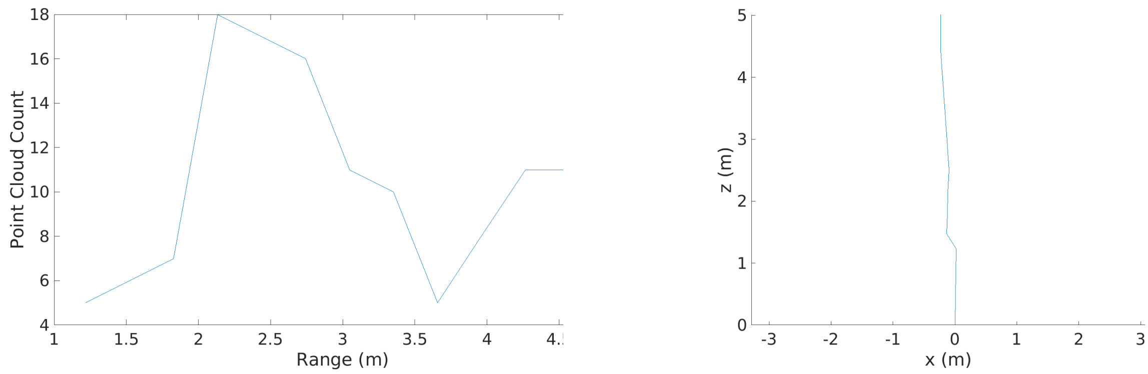


(a) Number of points in point cloud

(b) Target tracking VO path

Figure 6.37: Integral refocusing plenoptic odometry

sizes, with a minimum of six points, is given in Figure 6.38a. The small number of points in these point clouds makes removing erroneous points difficult for RANSAC in addition to purely estimating the rotation and translation. This problem is exacerbated by the shrinking size of the sign as it moves further from the camera. While this problem does not exist in ego-motion estimation, its parallel for typical ego-motion visual odometry is features that are out of range.



(a) Number of points in point cloud

(b) Target tracking VO path

Figure 6.38: FFT refocusing plenoptic odometry

Chapter 7

Future Work

This dissertation has shown plenoptic ranging for camera localization using perspective and refocused imagery rendered from plenoptic imagery. This dissertation began with a description of the perspective, integral refocused imagery, and FFT refocused rendering algorithms and their implementation on three processing architectures. Then, the traditional stereo point cloud using plenoptic perspective images was discussed along with the multiperspective point cloud generation algorithm. The ranging calibration for both the integral and FFT-based algorithms were modelled and the ranging algorithm discussed. Finally, experimental ranging results were presented in Chapter 6.

Since any localization algorithm requires a low processing time, this dissertation also described the algorithm architecture and processing times for rendering perspective images and refocused images with a single core processor, a multiple core processor, and on a GPU. The perspective images could be rendered either from the radiance, which required a longer computation time, or from the plenoptic image itself, which avoided computing the radiance and resulted in lesser computation time. For the low computationally-intensive process of perspective imaging, the computation time on the GPU was dominated by the transfer time and as such was close to the multicore processing time of 5 ms. The integral refocusing algorithm had the fastest refocused image rendering time at 17 ms of the three methods due to the lengthy computation time of the 4-D Fourier transform of the FFT refocused method at 40 ms. The other steps of the FFT refocused method itself only required an additional 8 ms for a total time of 49 ms. However for successive refocused image renderings of a scene, the FFT method is more computationally efficient due to its Fourier slice capability, which allows for successive renderings at different α terms of less than a millisecond on the GPU.

Traditional ranging with stereo cameras requires knowledge of the rotation and translation between the two cameras. Any change in relative orientation or position of the two cameras requires a new extrinsic calibration, which is a time consuming process. However, the extrinsic properties of the plenoptic camera are a function of the physical properties of the camera and lens. These physical properties are found through an exhaustive extrinsic calibration and were modeled in this dissertation as the radii of two spheres which were used for calculating the rotation and translation between two virtual pinhole cameras. With a model for the geometry of all of the possible virtual cameras used for generating perspective images, any two virtual cameras can be chosen, resulting in their extrinsic properties being known.

While two perspective images rendered from the plenoptic image can be used in a traditional stereo camera ranging algorithm to find disparity, a more robust estimate of range was developed through redundant disparity measurements by exploiting the multiple perspective capabilities of the plenoptic camera. Each feature in an image can be ranged by a virtual stereo pair calculated from the (u, v) image coordinate space. With the automatic extrinsic calibration equation, each virtual stereo image pair can be rectified for ranging. With the redundant ranges, erroneous ranges from a stereo pair can be ignored.

Depth from focus ranging was realized using either the integral method or the FFT method for rendering refocused images. The refocusing method was more straightforward than the traditional stereo vision approach due to the lack of extrinsic camera calibration; however, a ranging calibration was needed to model the relationship between the refocus parameter α and the range. The refocus parameter α dictated the range of the refocused image; an in-focus feature at a specified α, then, was at the corresponding distance of the range model. Template matching was used to score different candidate values of α for the best match to the in-focus perspective image. Two methods were developed for finding this best match: automatic refocusing and the α sweep method. With the value of α, the range to the feature could be calculated.

Feature points in the rendered perspective images were tracked from frame to frame, and each of the four ranging methods were used to calculate the range for each point in a 3-D point cloud. These point clouds were used for determining the corresponding rotation and translation of a moving target from one timestep to the next timestep. The ranging capability of each algorithm was evaluated by estimating the range to a known target. Two lenses were used for estimating range: a 55mm lens and a 135mm lens. For the 55mm lens, the stereo ranging algorithm and the multiple perspective ranging algorithm ranged to about 1.5 meters. The two refocusing methods could range to about 3 meters. The 135mm lens resulted in greater range capability with the stereo and multiperspective algorithms ranging to about 5 meters. The integral refocusing algorithm ranged to 7 meters, while the FFT refocusing algorithm detected distances to 4.5 meters. Additionally, the rotation and translation of the target was tracked over time using a visual odometry approach with each of the four ranging methods.

7.1 Tradeoffs of the Plenoptic Camera for Vision Navigation

This dissertation has shown several tradeoffs between depth discrimination, performance, and computational efficiency through the exploration of several sizes of lens. The horizontal field of view of the 135mm lens at 16.05° is significantly smaller than the 55mm lens at 40.47°, which can result in difficulty tracking features. However, the ranging capability of the 135mm lens is greater than the 55mm lens as exemplified by the extrinsically calibrated baseline of 8.28mm for the 55mm and 27.39mm for the 135mm lens.

Another tradeoff is computation time and depth discrimination for the integral and FFT refocusing methods. The integral method provides the fastest computation time for rendering a single image, but the FFT refocusing method can quickly generate multiple refocused images from a single plenoptic image through the Fourier slice theorem. However, the Fourier slice theorem only captures a portion of the 4-D light field, which results in a shorter maximum ranging capability.

Perspective imagery has a low computational burden, and the processing time for rendering perspective images using CUDA is dominated by the transport time to the GPU. While rendering perspective images using OpenMP is just as fast as CUDA in the general case or faster in the integer image coordinate case, if other rendered image types such as a refocused image is needed an in algorithm (i.e. depth from focus) the radiance will already have been sent to the GPU. In this case, the perspective imaging loses that transport time cost. Additionally, if a series of perspective images need to be rendered such as for the multiperspective case, sending the radiance to the GPU need only be done once and the transport cost is again reduced. Clearly, the application needed for the rendered images must be considered when choosing the fastest platform for rendering images.

Larger lens sizes provides greater range capability at the expense of field of view. A decision on each of these tradeoffs must be made for a specific environment and application. Some environments, such as indoor, require a smaller range capability, and as such a smaller lens size such as the 80mm lens may be chosen. For outdoor environments and large rooms, a larger lens size such as the 135mm lens is needed to accurately capture the range to features. However, some applications, such as capturing a scene, may need a larger field of view. An application such as collision detection may not need more than a meter of ranging ability, but it may need a wide field of view. In this case, the 55mm lens may be chosen.

7.2 Future Work

This dissertation has discussed several ideas towards further study in plenoptic point cloud generation in vision navigation. Other lenses for exploring the field of view versus ranging capability could lead to a better understanding of the best lens size for navigation with features at a particular depth and scene. Another source for study is varying several of the options available on the experimental camera such as effect of focusing the camera at particular points on the plenoptic image.

Of particular note is the use of other plenoptic cameras for generating point clouds of a scene. Lytro has provided a plenoptic camera with greater resolution and depth capabilities [122]. These cameras, however, are designed for aesthetic commercial use rather than as navigational aids in machine vision. Any plenoptic camera with higher plenoptic image resolution will result in greater computation times, which complicates the real-time requirements for vision navigation. However, a greater resolution can provide greater rendered image resolution, angular resolution, and number of microlens. Additionally, by increasing the maximum virtual baseline of a rendered perspective pair through greater lens sizes or microlens size, the depth capability of a plenoptic camera can be increased. Similarly, the ability of the plenoptic camera to use different baselines through different (u, v) coordinates of virtual cameras allows for ranging at various depths. Narrower baselines can more easily find matched features in the near range, while wider baselines can better estimate range in the far range through greater disparity discrimination [128]. However, to take full advantage of this feature of plenoptic cameras, the maximum baseline of the virtual cameras would need to be significantly increased through the hardware design of the plenoptic camera.

Sensor fusion of the plenoptic cameras with other sensors is a promising, unexplored avenue of plenoptic camera research. Any vision navigation algorithm that takes advantage of point clouds, such as SLAM algorithms, structure-from-motion algorithms, and visual odometry, can incorporate the point clouds generated from the plenoptic camera. Of particular note is the sensor fusion of cameras and inertial measurement units (IMU) in order to take advantage of the complementary aspects of low frequency but accurate plenoptic measurements with high frequency but drifting inertial measurements [5]. Filter architectures for the plenoptic camera and IMU sensor fusion include a loosely coupled approach where the rotation and translation output of the plenoptic odometry is used as an input to a Kalman filter, or a tightly coupled approach where the 3-D feature points themselves are states in the Kalman filter framework. Additional states can be added for the online estimation of possibly unknown, uncalibrated parameters such as the focal lengths, distance between the microlens array and the sensor array, and the principal points.

A plenoptic camera specifically designed for a greater field of view, maximum stereo baseline, and resolution can better track features and depth for vision navigation. The experimental camera used in this dissertation was designed only for close range measurements. A plenoptic camera that is designed explicitly for long ranges will greatly increase its ranging and therefore navigation performance.

Finally, using a plenoptic camera rather than stereo camera will also remove the need for multiple camera setups that often require synchronized triggers for image capture. The smaller space and power requirements of plenoptic cameras are also useful in environments such as indoors and underground that do not need extensive ranging capabilities. Additionally, the exhaustive extrinsic calibration routines to compensate for small changes in the rotation and translation of the stereo camera is removed by the dependency of the geometry of the virtual cameras on the hardware inside of the plenoptic camera itself. With greater image sensor sizes, microlens arrays, and thus ranging ability, plenoptic cameras can replace multiple camera setups for generating point clouds in vision navigation.

References

[1] R. Ng, "Digital light field photography," Ph.D. dissertation, Stanford University, 2006.

[2] S. H. Fuller and L. I. Millett, "Computing performance: Game over or next level?" *Computer*, vol. 44, no. 1, pp. 31–38, Jan 2011.

[3] R. K. Cavin, P. Lugli, and V. V. Zhirnov, "Science and engineering beyond moore's law," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1720–1749, May 2012.

[4] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[5] P. Corke, J. Lobo, and J. Dias, "An introduction to inertial and visual sensing," in *The International Journal of Robotics Research*, vol. 26, no. 6, June 2007, pp. 519–535.

[6] A. Isaksen, L. McMillan, and S. J. Gortler, "Dynamically reparameterized light fields," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 297–306. [Online]. Available: http://dx.doi.org/10.1145/344779.344929

[7] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, Oct 1996.

[8] J. E. Cutting, "How the eye measures reality and virtual reality," *Behavior Research Methods, Instruments, & Computers*, vol. 29, no. 1, pp. 27–36, 1997. [Online]. Available: http://dx.doi.org/10.3758/BF03200563

[9] A.-T. Tsao, Y.-P. Hung, C.-S. Fuh, and Y.-S. Chen, "Ego-motion estimation using optical flow fields observed from multiple cameras," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, Jun 1997, pp. 457–462.

[10] G. Xu and Z. Zhang, *Epipolar Geometry in Stereo, Motion, and Object Recognition: A Unified Approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[11] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000. [Online]. Available: http://dx.doi.org/10.1109/34.888718

[12] J. Bouguet, "Camera calibration toolboxbox for matlab." July 2013, [Online]. Available: $http://www.vision.caltech.edu/bouguetj/calib_doc/index.html$.

[13] A. Richardson, J. H. Strom, and E. Olson, "Aprilcal: Assisted and repeatable camera calibration," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[14] D. Brown, "Decentering distortion of lenses," *Photogrammetric Engineering*, vol. 32, pp. 444–462, 1966.

[15] ——, "Close-range camera calibration," *Photogra*, vol. 37, pp. 855–866, 1971.

[16] H. Louhichi, T. Fournel, J. M. Lavest, and H. B. Aissia, "Self-calibration of scheimpflug cameras: an easy protocol," *Measurement Science and Technology*, vol. 18, no. 8, p. 2616, 2007. [Online]. Available: http://stacks.iop.org/0957-0233/18/i=8/a=037

[17] R. Y. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv ccamera and lenses," *IEEE Transactions on Robotics and Automation*, vol. 3, pp. 323–344, 1987.

[18] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, 1999, pp. 666–673 vol.1.

[19] R. I. Hartley, “Theory and practice of projective rectification,” *International Journal of Computer Vision*, vol. 35, no. 2, pp. 115–127, 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1008115206617

[20] D. Bradski and A. Kaehler, *Learning OpenCV*, 1st ed. O’Reilly Media, Inc., 2008.

[21] A. Gershun, “The light field,” *Journal of Mathematics and Physics XVIII*, pp. 51–151, 1936, translated by P. Moon and G. Timoshenko.

[22] 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, no. 2, pp. 99–106, Feb 1992.

[23] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, “The lumigraph,” in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 43–54. [Online]. Available: http://doi.acm.org/10.1145/237170.237200

[24] K. Lynch, T. Fahringer, and B. Thurow, “Three-dimensional particle image velocimetry using a plenoptic camera,” in *Aerospace Sciences Meetings*. American Institute of Aeronautics and Astronautics, Jan. 2012, pp. –. [Online]. Available: https://doi.org/10.2514/6.2012-1056

[25] T. Fahringer and B. Thurow, “Tomographic reconstruction of a 3-d flow field using a plenoptic camera,” in *Fluid Dynamics and Co-located Conferences*. American Institute of Aeronautics and Astronautics, Jun. 2012, pp. –. [Online]. Available: https://doi.org/10.2514/6.2012-2826

[26] R. Haynes, B. Brock, and B. Thurow, “Application of mhz frame rate, high dynamic range piv to a high-temperature, shock-containing jet,” in *Aerospace Sciences Meetings*. American Institute of Aeronautics and Astronautics, Jan. 2013, pp. –. [Online]. Available: https://doi.org/10.2514/6.2013-774

[27] S. Carpenter, "Depth estimation with a plenoptic camera," *Proceedings of the AIAA 2013 Southeastern Regional Student Conference*, 2013.

[28] W. A. Roberts and B. S. Thurow, "Correlation-based depth estimation with a plenoptic camera," *AIAA Journal*, vol. 55, no. 2, pp. 435–445, Dec. 2016. [Online]. Available: https://doi.org/10.2514/1.J055050

[29] T. C. Cha Zhang, "A survey on image-based rendering - representation, sampling and compression," *Signal Processing: Image Communication*, vol. 19, pp. 1–28, January 2004. [Online]. Available: https://www.microsoft.com/en-us/research/publication/survey-image-based-rendering-representation-sampling-compression/

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

[31] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/237170.237199

[32] A. Lumsdaine and T. Georgiev, "The focused plenoptic camera," in *2009 IEEE International Conference on Computational Photography (ICCP)*, April 2009, pp. 1–8.

[33] T. Georgiev, "New results on the plenoptic 2.0 camera," in *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, Nov 2009, pp. 1243–1247.

[34] G. Schultz, "Magnetic resonance imaging with nonlinear gradient fields: signal encoding and image reconstruction," 2013.

[35] F. J. Harris, "On the use of windows for harmonic analysis with the discrete fourier transform," in *Proc. IEEE*, 1978, pp. 51–83.

[36] R. Stroebel, *Photographic Materials and Processes*, L. Stroebel, J. Compton, I. Current, and R. Zakia, Eds. Focal Press, 1986.

[37] R. Gonzalez and R. Woods, *Digital Image Processing*, 2nd ed., M. J. Horton, Ed. Tom Robbins, 2001.

[38] J. Dongarra and F. Sullivan, "Guest editors introduction to the top 10 algorithms," *Computing in Science Engineering*, vol. 2, no. 1, pp. 22–23, Jan 2000.

[39] Y. Yang, F. Liu, M. Li, J. Jin, E. Weber, Q. Liu, and S. Crozier, "Pseudo-polar fourier transform based compressed sensing mri," *IEEE Transactions on Biomedical Engineering*, vol. PP, no. 99, pp. 1–1, 2016.

[40] C. Li and J. Guan, "Code acquisition method of gps signals based on wfta, pfa and cooley-tukey fft," in *2012 IEEE International Conference on Intelligent Control, Automatic Detection and High-End Equipment*, July 2012, pp. 119–122.

[41] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[42] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, Sep. 1992. [Online]. Available: http://www.worldcat.org/isbn/052143064X

[43] W. M. Gentleman and G. Sande, "Fast fourier transforms: For fun and profit," in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, ser. AFIPS '66 (Fall). New York, NY, USA: ACM, 1966, pp. 563–578. [Online]. Available: http://doi.acm.org/10.1145/1464291.1464352

[44] P. Duhamel and M. Vetterli, "Fast fourier transforms: A tutorial review and a state of the art," *Signal Process.*, vol. 19, no. 4, pp. 259–299, Apr. 1990. [Online]. Available: http://dx.doi.org/10.1016/0165-1684(90)90158-U

[45] I. J. Good, "The interaction algorithm and practical fourier analysis," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 361–372, 1958. [Online]. Available: http://www.jstor.org/stable/2983896

[46] S. Winograd, "On computing the discrete fourier transform," *Mathematics of Computation*, vol. 32, no. 141, pp. 175–199, 1978. [Online]. Available: http://www.jstor.org/stable/2006266

[47] J. I. Jackson, C. H. Meyer, D. G. Nishimura, and A. Macovski, "Selection of a convolution function for fourier inversion using gridding [computerised tomography application]," *IEEE Transactions on Medical Imaging*, vol. 10, no. 3, pp. 473–478, Sep 1991.

[48] D. P. Mitchell and A. N. Netravali, "Reconstruction filters in computer-graphics," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 221–228, Jun. 1988. [Online]. Available: http://doi.acm.org/10.1145/378456.378514

[49] M. Nilchian, J. P. Ward, C. Vonesch, and M. Unser, "Optimized kaiser bessel window functions for computed tomography," *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3826–3833, Nov 2015.

[50] R. Ferreira and N. Goncalves, *Fast and Accurate Micro Lenses Depth Maps for Multi-focus Light Field Cameras*. Cham: Springer International Publishing, 2016, pp. 309–319. [Online]. Available: https://doi.org/10.1007/978-3-319-45886-1_25

[51] D. G. Dansereau, I. Mahon, O. Pizarro, and S. B. Williams, "Plenoptic flow: Closed-form visual odometry for light field cameras," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4455–4462.

[52] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[53] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[54] T. Nakashima, S. Oshima, and A. Nakashima, "Implementation of the performance evaluation system for the ntp server," in *2003 IEEE Pacific Rim Conference on Communications Computers and Signal Processing (PACRIM 2003) (Cat. No.03CH37490)*, vol. 2, Aug 2003, pp. 828–831 vol.2.

[55] M. Hack, X. Meng, S. Froehlich, and L. Zhang, "Leap second support in computers," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Sept 2010, pp. 91–96.

[56] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[57] M. Abramowitz, *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated, 1974.

[58] F. Mueller, "Pthreads Library Interface," 1994. [Online]. Available: http://dx.doi.org/10.1.1.54.6163

[59] M. P. Forum, "Mpi: A message-passing interface standard," Knoxville, TN, USA, Tech. Rep., 1994.

[60] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," *SIGPLAN Not.*, vol. 30, no. 8, pp. 207–216, Aug. 1995. [Online]. Available: http://doi.acm.org/10.1145/209937.209958

[61] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, A. J. Miller, and M. Upton, "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal*, vol. 6, no. 1, Feb. 2002. [Online]. Available: http://download.intel.com/technology/itj/2002/volume06issue01/art01_hyper/vol6iss1_art01.pdf

[62] D. Steinkrau, P. Y. Simard, and I. Buck, "Using gpus for machine learning algorithms," in *Proceedings of the Eighth International Conference on Document Analysis and*

Recognition, ser. ICDAR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1115–1119. [Online]. Available: http://dx.doi.org/10.1109/ICDAR.2005.251

[63] M. Bose and V. Rajagopala, "Physics engine on reconfigurable processor - low power optimized solution empowering next-generation graphics on embedded platforms," in *2012 17th International Conference on Computer Games (CGAMES)*, July 2012, pp. 138–142.

[64] S. A. Manavski, "Cuda compatible gpu as an efficient hardware accelerator for aes cryptography," in *2007 IEEE International Conference on Signal Processing and Communications*, Nov 2007, pp. 65–68.

[65] R. M. Stallman and G. DeveloperCommunity, *Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3*. Paramount, CA: CreateSpace, 2009.

[66] M. Harris, "How to optimize data transfers in cuda c/c++," Online, December 2012. [Online]. Available: https://devblogs.nvidia.com/parallelforall/how-optimize-data-transfers-cuda-cc/

[67] Nvidia, "CUDA CUFFT Library," 2007.

[68] Y. Zhang and Y. Zhang, "Research on monocular visual odometry based on 3d-2d motion estimation," in *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, July 2016, pp. 349–353.

[69] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1052–1067, 2007.

[70] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 1449–1456.

[71] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[72] E. Rosten and T. Drummond, *Machine Learning for High-Speed Corner Detection*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443. [Online]. Available: http://dx.doi.org/10.1007/11744023_34

[73] J. Shi and C. Tomasi, "Good features to track," in *Proc. CVPR '94. IEEE Computer Society Conf Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[74] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2548–2555. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2011.6126542

[75] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2011.6126544

[76] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 778–792. [Online]. Available: http://dl.acm.org/citation.cfm?id=1888089.1888148

[77] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, June 2012, pp. 510–517.

[78] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ser. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 1150–. [Online]. Available: http://dl.acm.org/citation.cfm?id=850924.851523

[79] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2007.09.014

[80] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7–42, 2002.

[81] K. Konolige, "Small vision systems: Hardware and implementation," in *Robotics Research*, Y. Shirai and S. Hirose, Eds. Springer London, 1998, pp. 203–212. [Online]. Available: $http://dx.doi.org/10.1007/978-1-4471-1580-9_19$

[82] H. Hirschmueller, P. Innocent, and J. Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 229–246, 2002.

[83] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, p. 2001, 2001.

[84] M. Lin and C. Tomasi, "Surfaces with occlusions from layered stereo," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 8, pp. 1073–1078, 2004.

[85] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 7, pp. 787–800, 2003.

[86] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 3551–3558.

[87] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 930–943, Aug 2003.

[88] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International Journal of Computer Vision*, vol. 81, no. 2, p. 155, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11263-008-0152-6

[89] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992. [Online]. Available: http://dx.doi.org/10.1109/34.121791

[90] R. I. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[91] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *3DIM*, 2001.

[92] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *Int. J. Comput. Vision*, vol. 13, no. 2, pp. 119–152, Oct. 1994. [Online]. Available: http://dx.doi.org/10.1007/BF01427149

[93] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment - a modern synthesis," in *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ser. ICCV '99. London, UK, UK: Springer-Verlag, 2000, pp. 298–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=646271.685629

[94] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Simultaneous localization and mapping: Present, future, and the robust-perception age," *CoRR*, vol. abs/1606.05830, 2016. [Online]. Available: http://arxiv.org/abs/1606.05830

[95] V. Drazic, "Optimal depth resolution in plenoptic imaging," in *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, July 2010, pp. 1588–1593.

[96] E. M. Hall, B. S. Thurow, and D. R. Guildenbecher, "Comparison of three-dimensional particle tracking and sizing using plenoptic imaging and digital in-line holography," *Appl. Opt.*, vol. 55, no. 23, pp. 6410–6420, Aug 2016. [Online]. Available: http://ao.osa.org/abstract.cfm?URI=ao-55-23-6410

[97] E. M. Hall, D. R. Guildenbecher, and B. S. Thurow, "Uncertainty characterization of particle location from refocused plenoptic images," *Opt. Express*, vol. 25, no. 18, pp. 21 801–21 814, Sep 2017. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-25-18-21801

[98] D. G. Dansereau, "Plenoptic signal processing for robust vision in field robotics," Ph.D. dissertation, University of Sydney, 2013.

[99] D. Dansereau and L. Bruton, "Gradient-based depth estimation from 4d light fields," in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, vol. 3, May 2004, pp. III–549–52 Vol.3.

[100] F. Dong, S.-H. Ieng, X. Savatier, R. Etienne-Cummings, and R. Benosman, "Plenoptic cameras in real-time robotics," *The International Journal of Robotics Research*, vol. 32, no. 2, pp. 206–217, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/2/206.abstract

[101] C. Hahne, A. Aggoun, V. Velisavljevic, S. Fiebig, and M. Pesch, "Refocusing distance of a standard plenoptic camera," *Opt. Express*, vol. 24, no. 19, pp. 21 521–21 540, Sep 2016. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-24-19-21521

[102] N. Zeller, F. Quint, and U. Stilla, "From the calibration of a light-field camera to direct plenoptic odometry," *IEEE Journal of Selected Topics in Signal Processing*, vol. PP, no. 99, pp. 1–1, 2017.

[103] R. Ng, M. Levoy, M. Bredif, G. Duval, M. Horowitz, and P. Hanraham, "Light field photography with a hand-held plenoptic camera," Stanford University, Tech. Rep., 2005.

[104] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1362–1376, Aug 2010.

[105] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building rome in a day," in *2009 IEEE 12th International Conference on Computer Vision*, Sept 2009, pp. 72–79.

[106] S. K. Huang, C. C. Yang, and H. H. Chen, "Empirical reliability analysis of disparity-based autofocus," in *2016 IEEE International Conference on Image Processing (ICIP)*, Sept 2016, pp. 998–1001.

[107] Y. Yoshida, S. Shinohara, H. Ikeda, K. Tada, H. Yoshida, K. Nishide, and M. Sumi, "Control of lens position in auto-focus cameras using semiconductor laser range finder," in *[1991] Proceedings of the 34th Midwest Symposium on Circuits and Systems*, May 1991, pp. 395–398 vol.1.

[108] S.-K. Huang, C.-C. Yang, K.-T. Shih, and H. H. Chen, "Using disparity information for stereo autofocus in 3-d photography," *Electronic Imaging*, vol. 2016, no. 18, pp. 1–6, 2016. [Online]. Available: http://www.ingentaconnect.com/content/ist/ei/2016/00002016/00000018/art00008

[109] M. Subbarao and J. K. Tyan, "Selecting the optimal focus measure for autofocusing and depth-from-focus," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 864–870, Aug 1998.

[110] A. Gaziowski, P. liwiski, and P. Wachel, "A stochastic approach to contrast detection autofocusing. a 2d case analysis," in *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, Aug 2016, pp. 738–743.

[111] P. liwiski and P. Wachel, "A simple model for on-sensor phase-detection autofocusing algorithm," *Journal of Computer and Communications*, no. 1, 2013.

[112] J. P. Lewis, "Fast normalized cross-correlation," 1995.

[113] A. Mahmood and S. Khan, "Early termination algorithms for correlation coefficient based block matching," in *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, vol. 2. IEEE, 2007, pp. II–469–II–472.

[114] J. Kiefer, "Sequential minimax search for a maximum," *Proceedings of the American Mathematical Society*, vol. 4, no. 3, pp. 502–506, 1953. [Online]. Available: http://www.jstor.org/stable/2032161

[115] L. Michaelis and M. L. Menten, "Die kinetik der invertinwirkung," *Biochem. Z*, vol. 49, no. 333-369, p. 352, 1913.

[116] H. Lineweaver and D. Burk, "The determination of enzyme dissociation constants," *Journal of the American Chemical Society*, vol. 56, no. 3, pp. 658–666, 1934. [Online]. Available: http://dx.doi.org/10.1021/ja01318a036

[117] E. W. Weisstein. Least squares fitting–exponential. Online. From MathWorld–A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/LeastSquaresFittingExponential.html

[118] MATLAB 9.1. Natick, Massachusetts, United States.

[119] *BOBCAT Camera Hardware User's Manual*, IMPERX, 6421 Congress Ave., April 2016. [Online]. Available: http://www.imperx.com/wp-content/uploads/Member/Cameras/Bobcat_Gen2/Bobcat_User_Manual__2_1_0.pdf

[120] *Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers*, Automated Imaging Association Std., Rev. 1.1, January 2004.

[121] H. Hsieh, "Express card interface adapter," Aug. 7 2007, uS Patent 7,254,664. [Online]. Available: https://www.google.com/patents/US7254664

[122] *Lytro ILLUM User Manual*, Lytro, Inc., 2015. [Online]. Available: https://s3.amazonaws.com/lytro-corp-assets/manuals/english/illum_user_manual.pdf

[123] C. Hahne, A. Aggoun, S. Haxha, V. Velisavljevic, and J. C. J. Fernndez, "Baseline of virtual cameras acquired by a standard plenoptic camera setup," in *2014 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, July 2014, pp. 1–3.

[124] E. Hall, T. Fahringer, D. Guildenbecher, and B. Thurow, "Volumetric calibration of a plenoptic camera," submitted to Applied Optics.

[125] A. Jennings. (2013, July) Sphere fit (least squared). Online. MathWorks FIle Exchange. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/34129-sphere-fit--least-squared-

[126] E. W. Weisstein. Circular segment. Online. From MathWorld–A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/CircularSegment.html

[127] O. Rodrigues, "Des lois gometriques qui regissent les dplacements d' un systme solide dans l' espace, et de la variation des coordonnes provenant de ces dplacement considres indpendant des causes qui peuvent les produire," *J. Math. Pures Appl.*, no. 5, pp. 380–440, 1840.

[128] D. Gallup, J. M. Frahm, P. Mordohai, and M. Pollefeys, "Variable baseline/resolution stereo," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[129] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan 1998, pp. 839–846.

Appendix A

Double Precision Execution Tables

Table A.1: CUDA Double Precision Data Structure Sizes

Data Structure Size	**(Bytes)**
Plenoptic Image	32170240
Radiance (Integral)	111288320
Radiance (FFT)	222576640
Image (Perspective and Integral)	217360
Image (FFT)	869440

Table A.2: Double Precision Serial Times for Image Rendering

Algorithm	**(s)**	
	Integer	General
Perspective from Radiance	0.2839	0.2896
Perspective from Plenoptic	0.003877	.02585
Integral Refocused	0.1429	0.9767
FFT Refocused	0.7319	0.7441

Table A.3: Double Precision Multicore Processing time for Generating Radiance

Radiance Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Non-Shifted	287.9	177.0	139.3	114.5	95.8	80.4	70.4	61.3
Shifted	286.8	174.1	138.3	116.3	100.5	85.0	73.6	65.0

Table A.4: Double Precision Multicore Processing time for Generating Perspective Image from Radiance

Perspective Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
General	6.718	3.749	2.772	2.211	1.828	1.51	1.324	1.184
Integer	2.139	1.294	0.9990	0.819	0.725	0.680	0.615	0.660

Table A.5: Double Precision Multicore Processing time for Generating Perspective Image from Plenoptic Image

Perspective Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
General	25.23	18.67	13.24	9.891	7.968	6.843	6.011	6.123
Integer	5.661	3.024	2.11	1.779	1.618	1.555	1.444	1.294

Table A.6: Double Precision Total Multicore Processing time for Generating Perspective Image from Radiance

Perspective Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Radiance	287.9	177.0	139.3	114.5	95.8	80.4	70.4	61.3
General	6.718	3.749	2.772	2.211	1.828	1.51	1.324	1.184
Total	294.6	180.7	142.1	116.7	97.63	81.91	71.72	62.48
Radiance	287.9	177.0	139.3	114.5	95.8	80.4	70.4	61.3
Integer	2.139	1.294	0.9990	0.819	0.725	0.680	0.615	0.660
Total	290.0	178.3	140.3	115.3	96.53	81.08	71.02	61.96

Table A.7: Double Precision Multicore Processing time for Generating Integral Refocused Image

Refocused Timing	**(s)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
General	1.118	0.683	0.524	0.438	0.371	0.313	0.270	0.237
Integer	0.210	0.143	0.114	0.091	0.075	0.064	0.056	0.050

Table A.8: Double Precision Processing time for Generating Refocused Image - Region of Interest

Timing	**(ms)**									
Scale	*.05*	*.10*	*1.5*	*.20*	*.25*	*.30*	*.35*	*.4*	*.45*	*.50*
Parallel	11.7	23.76	33.29	47.26	60.05	69.01	84.47	96.2	105.3	117.7
Parallel (Integer)	2.383	4.979	7.346	9.972	12.58	14.94	17.52	19.88	22.49	25.02
Serial	37.37	79.29	117.4	162.0	202.7	240.7	283.1	321.3	364.0	406.1
Serial (Integer)	6.725	14.19	20.91	28.36	35.81	42.51	50.01	56.69	64.18	71.65
Scale	*.55*	*.60*	*.65*	*.70*	*.75*	*.80*	*.85*	*.90*	*.95*	*1.0*
Parallel	134.1	144.6	152.5	171.8	183.1	188.5	205.3	220.1	224.3	235.3
Parallel (Integer)	27.38	29.94	32.27	34.83	37.40	39.71	42.28	44.60	47.10	49.67
Serial	455.3	486.6	524.8	567.4	609.6	647.7	690.2	728.4	771.3	807.5
Serial (Integer)	78.37	85.83	92.49	100.0	107.4	114.1	121.5	128.3	135.8	143.2

Table A.9: Double Multicore Processing time for Generating Integral Refocused Image

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Non-Shifted Radiance	287.9	177.0	139.3	114.5	95.8	80.4	70.4	61.3
Integral Refocusing	1118	683	524	438	371	313	270	237
Total	1406	860	663.3	552.5	466.8	393.4	340.4	298

Table A.10: Double Precision Multicore Processing time for Generating Integral Refocused Image - Integer

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Non-Shifted Radiance	287.9	177.0	139.3	114.5	95.8	80.4	70.4	61.3
Integral Refocusing	210	143	114	91	75	64	56	50
Total	497.9	320	253.3	205.5	170.4	144.4	126.4	111.3

Table A.11: Double Precision Multicore Processing time for Generating FFT Refocused Image

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Shifted Radiance	290.0	237.0	136.7	127.8	104.1	88.1	79.64	66.34
4-D FFT	443.7	248.2	255.6	178.6	193.0	167.1	159.8	152.9
4-D Slice	2.499	2.043	1.372	1.094	0.868	0.748	0.64	0.590
2-D iFFT	1.958	1.328	0.8957	1.014	1.19	1.308	1.067	1.184
Total	738.2	488.8	394.6	308.5	299.2	257.3	241.1	221.0

Table A.12: Double Precision Multicore Processing time for Generating FFT Refocused Image - $\alpha = 1$

Refocused Timing	**(ms)**							
Cores	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Shifted Radiance	284.9	232.7	132.1	124.	102.1	85.55	78.55	72.74
4-D FFT	442.7	245.	265.	184.8	199.8	176.3	169.5	159.4
4-D Slice	2.519	1.901	1.27	.9994	.8499	.7256	.6236	.6673
2-D iFFT	1.742	1.243	.9599	.8073	1.126	1.28	.9729	1.216
Total	731.9	480.8	399.3	310.6	303.9	263.9	249.6	234.0

Table A.13: CUDA Double Precision Processing time for Generating Perspective Image

CUDA Perspective Timing	**(ms)**			
	Radiance		**Plenoptic**	
	Integer	*General*	*Integer*	*General*
Plenoptic Image Transport	4.34	4.34	4.455	4.455
Radiance Generation	5.746	5.746	0	0
Perspective Image Rendering	0.0773	0.333	0.04959	0.4106
Perspective Image Transport	0.0114	0.0114	0.1067	0.1067
Total	10.1747	10.4304	4.611	4.9723

Table A.14: CUDA Double Precision Processing Time for Generating Integral Refocused Image

Refocused Timing	**(ms)**	
	Integer	*General*
Plenoptic Image Transport	4.381	4.359
Radiance Generation	5.757	5.754
Refocused Image Rendering	27.99	40.62
Refocused Image Transport	0.01147	0.01168
Total	38.14	50.745

Table A.15: Double Precision Processing Time for Slicing 4-D - Block Size 8

Fourier Slice Timing	**(ms)**	
	General	Integer
Fourier Slice	0.4021	0.1775

Table A.16: Double Precision Processing Time for Slicing 4-D - Variable Thread Block Sizes

Slice Timing	**(ms)**							
Thread Block Size	*1×1*	*2×2*	*3×3*	*4×4*	*5×5*	*6×6*	*7×7*	*8×8*
Fourier Slice	4.277	2.162	1.061	.6522	.4849	.6126	.5037	.4021
Thread Block Size	*9×9*	*10×10*	*11×11*	*12×12*	*13×13*	*14×14*	*15×15*	*16×16*
Fourier Slice	.5676	.6105	.57	.6676	.7811	1.033	1.038	1.067

Table A.17: Double Precision Processing Time for FFT-based Rofocusing

Fourier Slice Timing	**(ms)**	
	General	Integer
Plenoptic Image Transport	4.305	4.419
Shifted Radiance	6.025	6.05
4-D FFT	98.79	98.55
Fourier Slice	0.4021	0.1775
2-D inverse FFT	0.2856	0.2862
Rendered Image Transport	0.1826	0.1826
Total	110	109.6

Table A.18: Double Precision Processing Time

Execution Times	**(ms)**	
	Integer	General
Perspective Imaging		
Serial	3.877	25.85
OpenMP	1.294	6.123
CUDA	4.611	4.972
Integral Refocusing		
Serial	142.9	976.7
OpenMP	50.0	237.0
CUDA	38.14	50.745
FFT Refocusing		
Serial	731.9	744.1
OpenMP	234	221
CUDA	110	109.6

Appendix B

Zero Padded Execution Table (ms)

50	45	71	48	47	47	77	51	65	52	49	59	69	50	79	51	59	52	88	59	59	56
46	42	68	44	44	43	72	48	61	47	45	55	65	46	75	47	55	47	84	54	55	52
50	46	72	48	47	47	77	52	66	52	49	59	69	50	79	51	60	52	89	59	60	57
60	55	82	58	57	57	87	63	76	62	60	70	80	61	91	62	71	63	100	70	71	68
50	45	72	48	47	47	77	52	66	51	49	59	69	50	80	52	59	52	89	59	59	56
52	47	74	50	49	49	79	54	68	53	51	61	72	52	82	53	62	54	91	61	62	59
79	66	80	67	81	69	82	83	84	84	75	85	86	74	87	88	88	76	89	90	91	91
48	43	70	46	45	44	75	50	64	49	47	57	67	48	78	49	57	49	87	56	57	54
80	67	81	68	82	69	83	84	85	85	76	86	87	74	88	89	90	76	91	91	92	92
52	47	74	50	49	48	80	54	68	53	51	61	72	52	82	53	62	54	92	61	62	59
53	48	75	50	50	50	81	55	69	54	52	62	73	53	83	54	63	55	93	62	63	60
50	45	73	48	46	46	78	51	66	51	48	59	70	50	80	51	59	51	90	59	59	56
54	50	77	52	51	51	83	56	71	56	53	64	75	55	86	56	64	57	96	64	65	62
54	49	77	52	51	51	83	56	71	55	53	64	75	54	86	55	64	56	96	64	65	61
82	69	84	70	85	71	86	87	87	88	79	89	90	77	91	92	92	79	93	95	95	96
68	63	92	66	65	66	98	71	86	71	68	80	91	70	102	72	81	73	113	80	82	78
62	56	85	59	58	58	91	64	79	63	61	72	83	63	95	64	73	65	105	73	74	70
51	46	75	49	48	48	80	53	68	52	50	61	72	51	83	52	61	53	93	61	61	58
57	52	81	55	54	54	87	59	74	59	56	67	78	58	90	59	68	59	100	67	68	65
51	46	75	49	48	48	81	53	68	52	50	61	72	51	84	52	61	53	94	61	61	58
84	70	85	71	87	72	88	88	89	90	80	91	92	78	93	94	94	81	95	96	97	98
61	55	85	58	57	57	90	62	78	62	60	71	82	61	94	62	72	63	105	71	72	69

Appendix C

Stereo Odometry Backend Simulation

One of the most well-developed vision navigation systems is stereo odometry. In this section, a "stereo camera" is defined as both a traditional stereo camera pair modeled as two pinhole cameras as well as two perspectives from a plenoptic camera that are modeled as two virtual pinhole cameras. The performance of the stereo pipeline backend described in Section 5.2.2 is highly dependent on the extrinsics and intrinsic parameters of the cameras. Two significant performance differences between a traditional stereo camera pair and two rendered perspective images from a plenoptic camera are the geometry of the modeled pinhole cameras (extrinsic parameters) and the intrinsic parameters (camera matrix). To explore the performance of the stereo odometry back end using a traditional stereo pair and two perspective images from a plenoptic camera, a simulation was developed that calculates simulated corresponded feature points from a static feature point field to ensure that no image processing errors from the front end are present. Since no actual images will be used, feature points will be simulated by projecting a point cloud onto the left and right image planes for each time step along a path. The true path is the output of a navigation solution, as shown in Figure C.1. The task of the back end of the stereo camera algorithm is to take the corresponded features from the left and right images of a stereo camera, generate the point cloud from the disparity, and calculate the rotation and translation from one time step to the next.

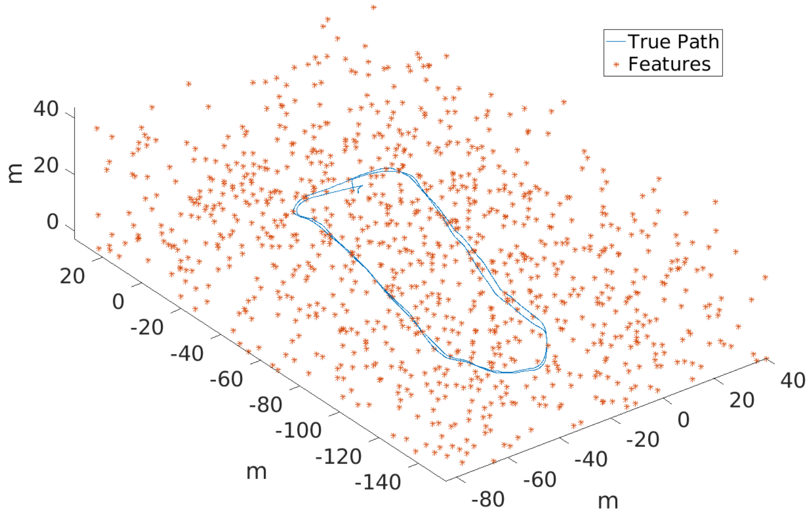


Figure C.1: True path and features

C.1 Coordinate Frames and Nomenclature

This simulation will use three coordinate frames: North-East-Down (N), body frame (B), and the camera frame (C). The body frame extends the x-axis forward, the y-axis to the right, and the z-axis down. The camera frame consists of the x-axis to the right, the y-axis down, and the z-axis forward. Rotations from one frame to another frame is notated using subscripts. For example, a rotation from the North-East-Down coordinate frame to the body frame is R_{NB} for a direction cosine matrix, ea_{NB} for an euler angle, and q_{NB} for a quaternion. Vectors are described by the tail of the vector to the head of the vector while the coordinate frame is designated as the subscript. A vector from point O to point P in the North-East-Down frame, then, is notated by $\overrightarrow{OP}_N$. Note that the camera frame is designated as a subscript, whereas the camera position is designated in the vector.

In this simulation, the rotation from the camera frame to the body frame, R_{CB}, is assumed to be known, which includes the axes rotation as well as any inherent rotation of the camera with respect to the body. The camera matrices for both the left and right cameras and the baseline between the left and right cameras are known as well.

C.2 Setting up the Simulation

Since the input for the backend of the visual odometry algorithm is the corresponded features, the corresponded features must be generated first from a point cloud in world coordinates at each point on the truth path. The correspondence for each feature is carried out by assigning a feature identification number to each feature and monitoring each feature from frame to frame. Note that there is no image processing matching done from frame to frame. The correspondence is conducted by ensuring that the feature point identification number in the last frame matches the feature point identification number for the next frame.

C.2.1 True Point Cloud

The feature points represented edges of real world objects, so the true point cloud will be generated in the world frame for this simulation (the North-East-Down frame). For this simulation, the true point cloud is generated as a uniform distribution over the maximum and minimum positions of the true path for all three dimensions plus a small buffer, b_{uf}. The true feature point locations, F_N^{true}, in the NED coordinate frame are calculated from the true camera locations, C_N^{true}, as shown in Equation (C.1)

$$F_N^{true} = \mathcal{U}(min(C_N^{true}) + b_{uf}, max(C_N^{true}) + b_{uf}) \tag{C.1}$$

where $\mathcal{U}(a, b)$ is the uniform distribution from a to b. Figure C.1 shows one example of a generated point cloud in the world frame.

C.2.2 True Point Cloud to Image Projection

With the feature points in the NED coordinate, F_N^{true}, and the camera positions in the NED coordinate, C_N^{true}, the vector from the camera positions to the feature points is shown in Equation (C.2)

$$\overrightarrow{CF}_N^{true} = \overrightarrow{OF}_N^{true} - \overrightarrow{OC}_N^{true} \tag{C.2}$$

where O designates the origin of the NED coordinate. The vector from the camera position to the feature point, $\overrightarrow{CF}_N^{true}$, must be rotated into the camera frame using the known camera to body rotation and the true rotation from the body frame to the NED coordinate.

$$\overrightarrow{CF}_C^{true} = R_{CB} R_{BN}^{true} \overrightarrow{CF}_N^{true} \tag{C.3}$$

The left camera serves as the main camera frame for this simulation.

$$\overrightarrow{C^L F}_C^{true} = \overrightarrow{CF}_C^{true} \tag{C.4}$$

The right camera is offset from the left camera by

$$\overrightarrow{C^R F}_C^{true} = \overrightarrow{C^L F}_C^{true} - C_{offB} \tag{C.5}$$

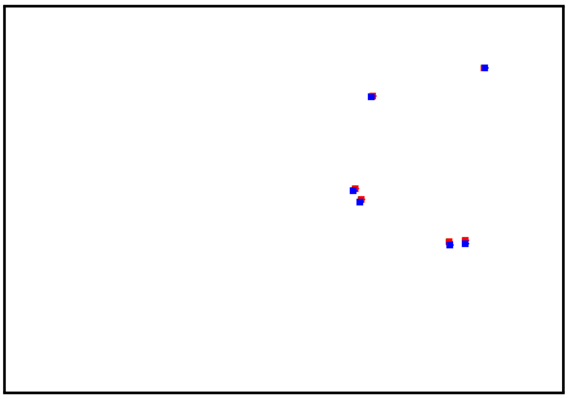
where C_{offB} is the vector offset in the body frame from the left camera to the right camera, e.g.

$$C_{offB} = \begin{bmatrix} 0 \\ b \\ 0 \end{bmatrix} \tag{C.6}$$

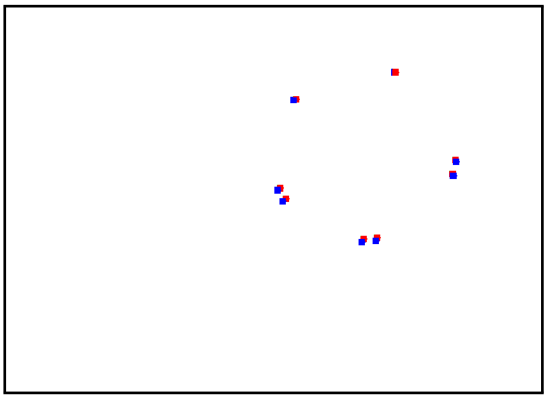
for a horizontal offset where b is the baseline. For both the right and left camera, each feature whose z-coordinate in the camera frame is negative must lie behind the camera and are thus ignored. Features in front of the camera are projected onto the image plane as shown in Equation (C.7)

$$p(x, y) = M_C \frac{\overrightarrow{C^L F}_C}{\overrightarrow{C^L F}_C[2]} \tag{C.7}$$

where M_C is the intrinsic matrix of the camera and $p(x, y)$ is the pixel coordinate. The fraction in this equation moves the three dimensional coordinates into a homogeneous vector by dividing by $\overrightarrow{C_L F}_C[2]$. Any calculated pixel locations $p(x, y)$ that are outside of the image bounds are ignored.



(a) Left image of target



(b) Right image of target

Figure C.2: Stereo pair when viewing simulated circular target

Figure C.2 shows the simulated feature points for the left and right camera images. The blue dots are the current frame features, while the red dots are the corresponded features from that last frame pair. Figure C.3 shows the path and camera view at the moment the images of Figure C.2 were generated. A feature target was created which has five points arranged in a circular pattern with two features side-by-side on three of the points in order to ensure the correct angle. While the path leads directly through the 8-feature point circular target, notice that the left camera view, represented as the green fulcrum, is rotated slightly to the left. This rotation is reflected in the generated images in Figure C.3 through capturing only half of the circle of points.

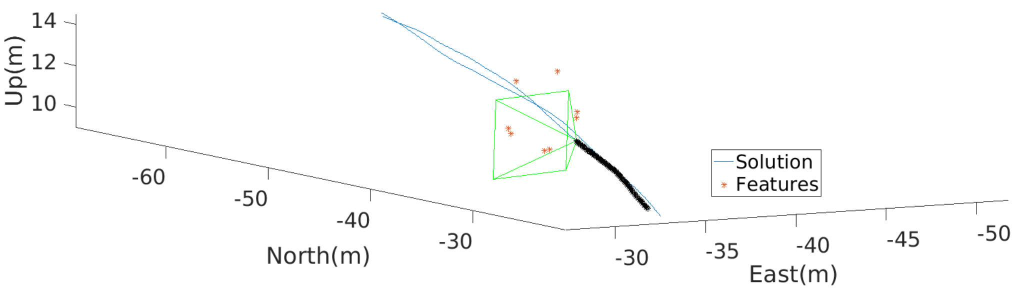


Figure C.3: Example path for circular target

C.3 Stereo Camera Backend Processing

Section C.2 showed how features were generated for a simulated stereo image pair using the output of a navigation filter solution as truth and simulated static field of points. Assuming a known baseline between the simulated stereo pair, a rotation from the camera frame to the body frame, and the camera intrinsics, an estimated rotation and translation can be found. Therefore, this process of extracting that estimated rotation and translation from the feature points will now be described, followed by a comparison of these estimated results to truth.

C.3.1 Point Cloud Generation

With the left and right images generated from feature points, the disparities for the corresponded feature points can be calculated. Since only features are present, the disparity will be calculated directly from the left and right camera feature points themselves using the distance formula in order to facilitate any angle for the stereo pair orientation.

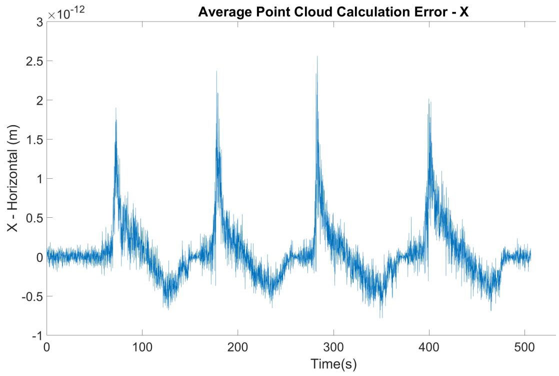
$$d = \sqrt{(x_L - x_R)^2 + (y_L - y_R)^2} \tag{C.8}$$

With the disparity, d, and intrinsic camera matrix, M_C, each feature point can be projected to their 3D point using the Q matrix of Equation (C.9)

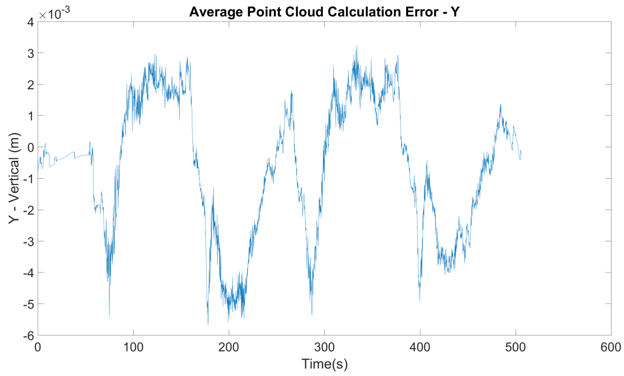
$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f_x \\ 0 & 0 & \frac{-1}{||C_{off}||} & \frac{c_x - c_x'}{-||C_{off}||} \end{bmatrix} \tag{C.9}$$

through the projection equation

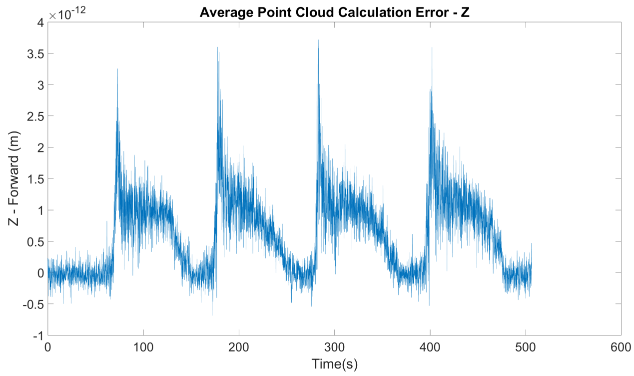
$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \tag{C.10}$$



(a) Projection error - X (camera frame)



(b) Projection error - Y (camera frame)



(c) Projection error - Z (camera frame)

Figure C.4: Point cloud projection error

where the 3-D points can be converted from their homogeneous counterparts through Equation (C.11).

$$\begin{bmatrix} \frac{X_h}{W} \\ \frac{Y_h}{W} \\ \frac{Z_h}{W} \\ \frac{W}{W} \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{C.11}$$

Figure C.4 shows small projection error due to perfectly known intrinsic and extrinsic camera properties (since both camera properties were used to generate the feature points), sub-pixel disparities, and no distortion. This error, in the NED coordinate frame, shows at most an average of millimeter error in the y direction and significantly less error in the other two axes. With this level of accuracy, high confidence can be taken for calculating the change in rotation and translation from frame to frame using SVD.

C.3.2 Rotation and Translation

The previous section has shown how to calculate the 3-D point clouds from a pair of stereo cameras at a point in time. Section 5.2.2.3.2 described the procedure for determining the rotation, ΔR_C, and translation, ΔT_C, through the SVD approach. By rotating the first point cloud A with the estimated change in rotation and translation, the average minimization error for the rotated and translated point cloud can be found.

$$error = \frac{1}{L}\sum_{i=0}^{L} \|\Delta R_C * A_n + \Delta T_C - B_n\|^2 \tag{C.12}$$

For calculating an estimated path using the estimated change in rotation and translation, an initial rotation and translation with respect to the NED coordinate frame must be known. Assuming the path starts at the origin of the NED frame at a known rotation R_{BN}, the position estimates from the stereo visual odometry algorithm can be found in the NED coordinate frame. Note that the rotation from the camera frame to the NED coordinate frame, R_{CN}^{est} will be used to update the rotation over time, while the position vector from the origin of the NED coordinate frame to the estimated camera location, $\overrightarrow{O_{CN}^{true}}$, will be updated in the NED coordinate frame.

C.3.3 Initial Conditions

An initial translation and rotation is needed since the visual odometry algorithm is inherently a relative navigation estimate. For these simulations, the known truth at the start of the run will be used for initialization.

$$R_{0CN}^{est} = R_{CB} * R_{BN}^{truth} \tag{C.13}$$

The simulation also assumes that the path begins at the origin of the NED frame.

$$\overrightarrow{O_{C0N}^{true}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{C.14}$$

C.3.4 Estimated Position

With the change in rotation, ΔR_C, and the change in translation, ΔT_C, in the camera frame, the estimated position can be found. First, the accumulated rotation from frame to frame must be updated for the new time.

$$R_{CN}^{est} = \Delta R_C R_{CN}^{est} \tag{C.15}$$

The estimated position can then be updated to the current time with the rotation matrix, R_{CN}^{est}.

$$\overrightarrow{O}_{CN}^{est} = \overrightarrow{O}_{CN}^{est} + R_{CN}^{est} \Delta T_C \tag{C.16}$$

C.4 Simulation of a Traditional Stereo Camera

A simulation was run using a traditional stereo camera with a 9 cm baseline and known intrinsic matrix, M_C, over the path previously seen in Figure C.1. This run is a base case run where no sources of error (apart from numerical errors) are added. Two methods will be used to evaluate the estimated rotation and translation from the 3-D point cloud alignment in this simulation: comparison of the rotation and translation with truth and the position error compared with truth. The error of the position solution calculated using the equations from Section C.3.4 is as seen in Figure C.5. The error reaches a maximum of approximately 10 cm. As a check on the quality of the SVD solution, the error as described in Equation (C.12) is shown below in Figure C.6. The distance traveled for this run was 638.1677m. The final error is .0024 m, with a percent traveled error of .036266%. For 2-D, the final error is .002 m, which results in an error of .031962% of distance traveled. The small error in this simulation is due to very small error in the simulation. In a real system, the error will be much larger.

The output of the visual odometry algorithm are the change in rotation and translation from frame to frame. Figure C.7 shows the error between the truth and estimated change in rotation over the course of the data run in degrees. Note only is the error limited to less than a thousandth of a degree, the error that is present is unbiased. Figure C.8 shows the error in the

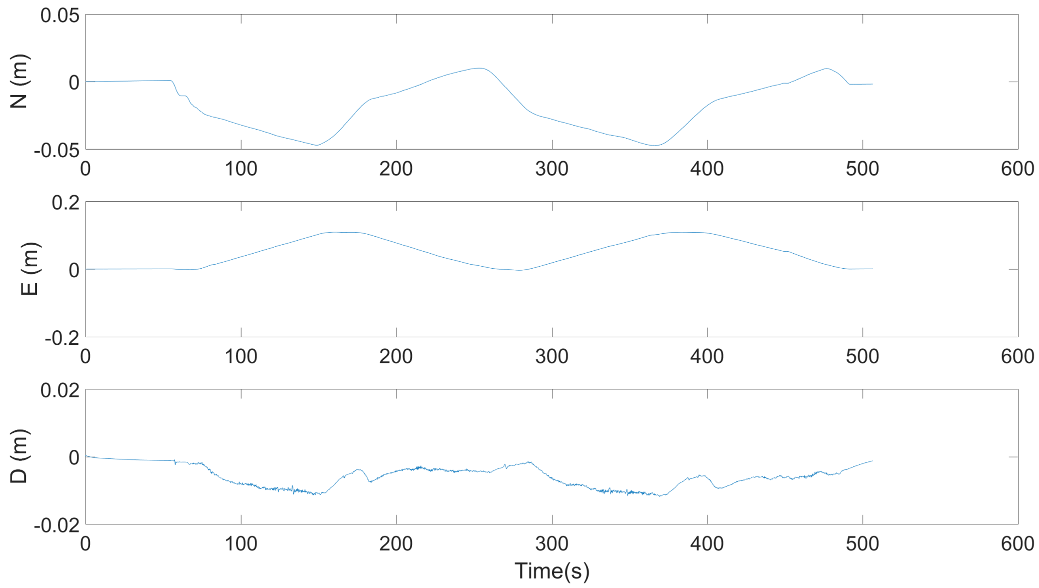


Figure C.5: Traditional Stereo: Error compared with Truth in the NED Frame

change in translation from frame to frame in the NED frame between the truth and the estimate of translation from visual odometry. Error in all three axes is less than a millimeter per frame. Unlike the change in rotation error, the errors in the East and North axes are slightly biased, which shows a small drift in the solution.

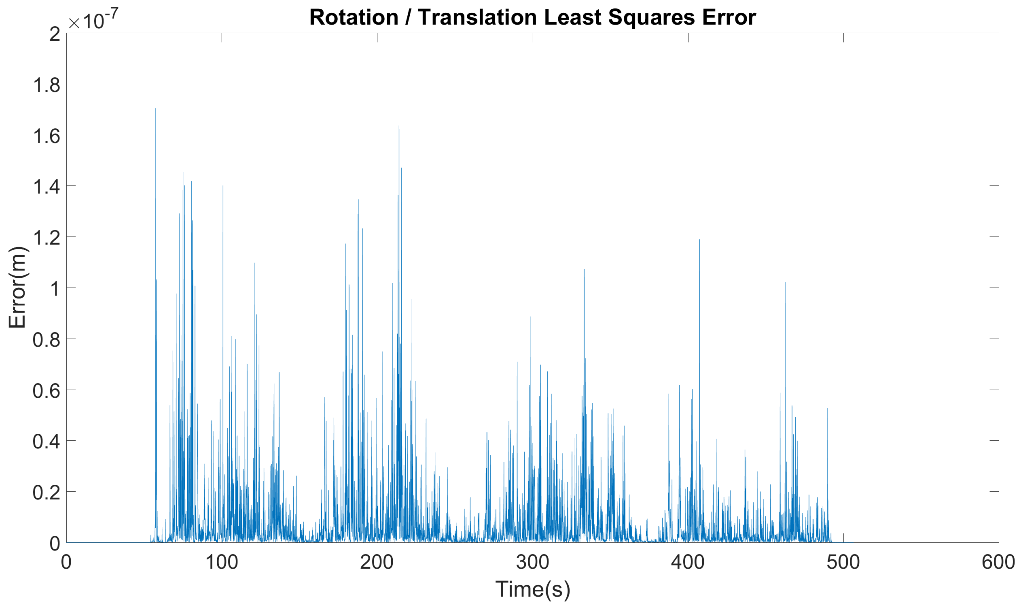


Figure C.6: Traditional Stereo: SVD fitting error

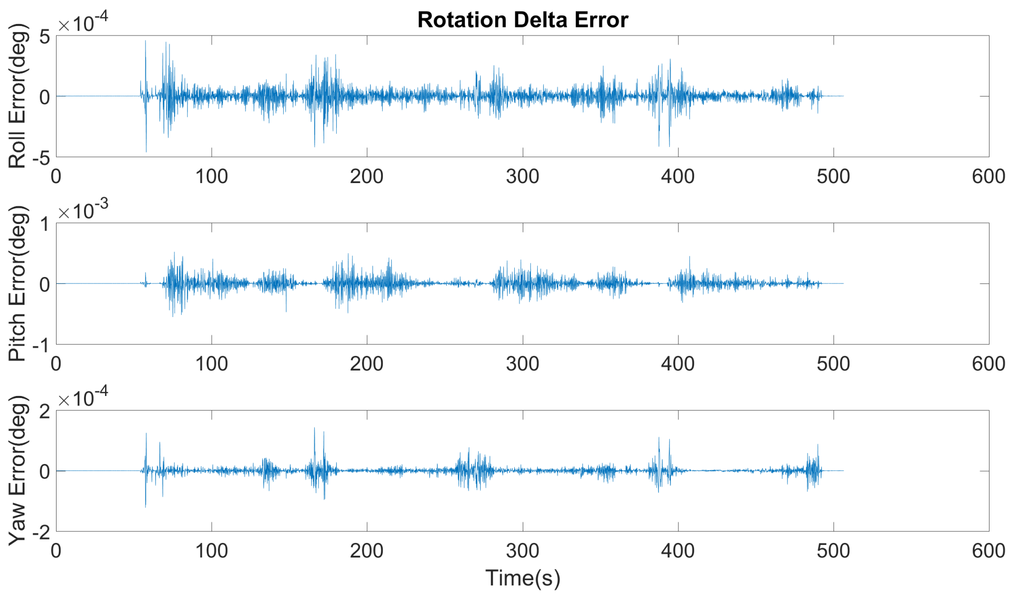


Figure C.7: Traditional Stereo Pair: Change in Rotation Error

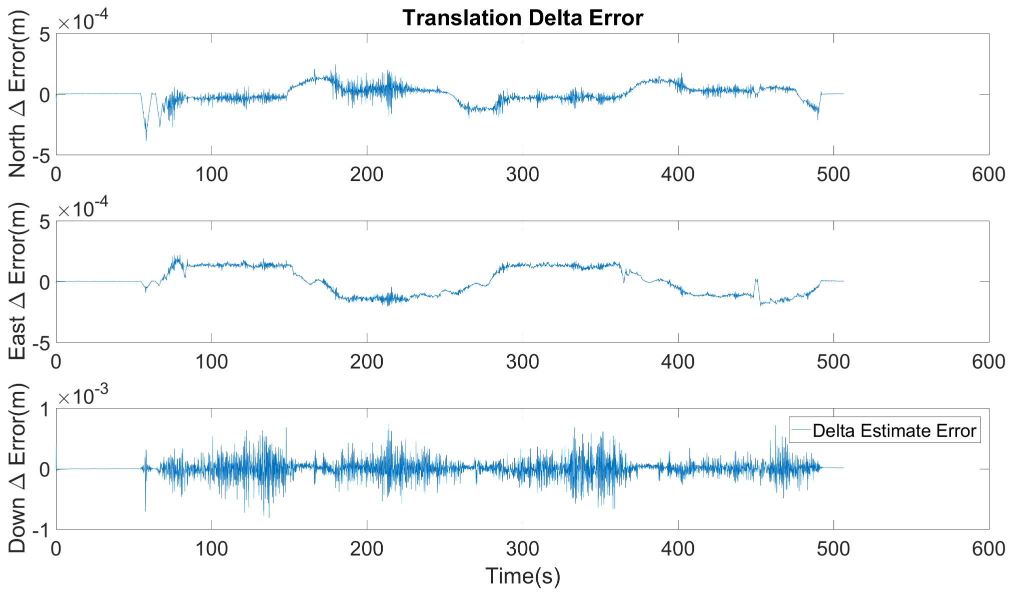


Figure C.8: Traditional Stereo Pair: Change in Translation Error

Appendix D

Refocusing Model Parameters

This appendix lists the model parameters calculated through the range calibration of Section 6.5.

D.1 Integral

D.1.1 55 mm lens

Table D.1 provides the parameters calculated through the regression methods described in Section 6.5.1.1. The horizontal offset added for the hyperbolic model reduces the c parameter drastically. The nonlinear hyperbolic fit has a very large offset, which is handled with the addition of the offsets in the other hyperbolic models. The b parameter of the non-linear exponential model is very large due to the location of the Z intercept of the model, which makes the exponential model undesirable for the 55 mm lens. The Michaelis-Menten model parameters between the least squares and non-linear regression approaches are similar, which shows that the linear model is a good approximation for the regression.

Table D.1: Integral Method Model Parameters for 55 mm

Model	**Parameter**	**Value**
$\frac{c}{\alpha^b}$ (linear)	b	4.2768
	c	19.1767
$\frac{c}{\alpha^b}$	b	10.8898
	c	598.7955
$a + \frac{c}{\alpha^b}$	a	0.53766
	b	15.8174
	c	5517.1348
$a + \frac{c}{(\alpha-d)^b}$	a	0.52338
	b	8.4162
	c	0.65963
	d	0.77412
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	V_{max}	0.18698
	K_m	-1.5189
$\frac{V_{max}\alpha}{K_m+\alpha}$	V_{max}	0.17386
	K_m	-1.534
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	V_{max}	0.12905
	K_m	-2.1709
	d	-0.64015
$be^{c\alpha}$ (linear)	b	69.1417
	c	-2.0858
$a + be^{c\alpha}$	a	0.55287
	b	13861550.6035
	c	-9.5484

D.1.2 80 mm lens

Table D.2 provides the parameters calculated through the regression methods described in Section 6.5.1.2. The hyperbolic least squares model, the nonlinear hyperbolic model, and the nonlinear least squares model with a vertical offset all have very similar models, where the vertical offset is close to zero with $a = .007$. After introducing the horizontal offset, the parameters change significantly with the shift of the asymptote. The Michaelis-Menton models without an offset have poor fits. The model with the offset results in large parameters for the K_m and d parameters.

Table D.2: Integral Method Model Parameters for 80 mm

Model	Parameter	Value
$\frac{c}{\alpha^b}$ (linear)	b	2.095
	c	2.4021
$\frac{c}{\alpha^b}$	b	2.1458
	c	2.4515
$a + \frac{c}{\alpha^b}$	a	0.0069956
	b	2.1595
	c	2.4462
$a + \frac{c}{(\alpha-d)^b}$	a	0.090968
	b	3.5826
	c	15.8888
	d	-0.70446
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	V_{max}	2.0033e-07
	K_m	-4194303.1769
	d	-4194302.5132
$be^{c\alpha}$ (linear)	b	4.654
	c	-0.97247
$a + be^{c\alpha}$	a	0.19979
	b	15.8623
	c	-1.9669

D.1.3 135 mm lens

Table D.3 in provides the parameters calculated through the regression methods described in Section 6.5.1.3. The Michaelis-Menten model with an offset that succeeded in capturing the curve has large parameters values for K_m and the horizontal offset parameter d.

Table D.3: Integral Method Model Parameters for 135 mm

Model	Parameter	Value
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	V_{max}	1.4848
	K_m	-0.04711
$\frac{V_{max}\alpha}{K_m+\alpha}$	V_{max}	2.6131
	K_m	-0.028872
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	V_{max}	2.087e-06
	K_m	-2097146.8296
	d	-2097147.9514
$be^{c\alpha}$ (linear)	b	3.4355
	c	-0.3755
$a + be^{c\alpha}$	a	0.36264
	b	3.7518
	c	-0.93645

D.2 FFT

D.2.1 55 mm lens

Table D.4 provides the parameters calculated through the regression methods described in Section 6.5.2.1. The hyperbolic least squares model significantly differs from the parameters for the non-linear hyperbolic models, which shows the importance of the non-linear regression over least squares methods for this dataset. The non-linear hyperbolic regression model without the Z offset assumes an offset of $a = 0$ m, while the nonlinear model with the offset calculates that offset to be $a = 0.5$ m. The parameters of both of the exponential models are very large due to the location of the Z intercept of the model, which makes the exponential model undesirable for the 55 mm lens. The Michaelis-Menten model parameters between the least squares and non-linear regression approaches are similar.

Table D.4: FFT Method Model Parameters for 55 mm

Model	**Parameter**	**Value**
$\frac{c}{\alpha^b}$ (linear)	b	33.9046
	c	10.0109
$\frac{c}{\alpha^b}$	b	99.1211
	c	218.5065
$a + \frac{c}{\alpha^b}$	a	0.51048
	b	142.4329
	c	1212.85
$a + \frac{c}{(\alpha-d)^b}$	a	0.51045
	b	140.9661
	c	264.2458
	d	0.010762
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	V_{max}	0.028472
	K_m	-1.0323
$\frac{V_{max}\alpha}{K_m+\alpha}$	V_{max}	0.021293
	K_m	-1.0369
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	V_{max}	0.010929
	K_m	-2.0427
	d	-1.0058
$be^{c\alpha}$ (linear)	b	418084473782213.9
	c	-31.4442
$a + be^{c\alpha}$	a	0.51295
	b	1.549529452949633e+62
	c	-136.2373

D.2.2 80 mm lens

Table D.5 provides the parameters calculated through the regression methods described in Section 6.5.2.2. The hyperbolic least squares model, unlike the 55 mm lens, does not differ significantly from the parameters for the non-linear hyperbolic models. The non-linear hyperbolic regression model without the Z offset assumes an offset of $a = 0$ m, while the nonlinear model with the offset calculates that offset to be $a = 0.2$ m. The parameters of both of the exponential models are still large due to the location of the Z intercept of the model, but due to the shift of the dataset α values, the intercept is not as large.

Table D.5: FFT Method Model Parameters for 80 mm

Model	Parameter	Value
$\frac{c}{\alpha^b}$ (linear)	b	15.2474
	c	1.8948
$\frac{c}{\alpha^b}$	b	22.1219
	c	2.3733
$a + \frac{c}{\alpha^b}$	a	0.2
	b	27.5554
	c	2.2477
$a + \frac{c}{(\alpha-d)^b}$	a	0.20016
	b	27.9357
	c	3.3113
	d	-0.01397
$\frac{V_{max}\alpha}{K_m+\alpha}$	V_{max}	0.056972
	K_m	-0.97774
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	V_{max}	1.1684e-07
	K_m	-524288.1545
	d	-524287.1785
$be^{c\alpha}$ (linear)	b	2012244.3162
	c	-13.9113
$a + be^{c\alpha}$	a	0.21287
	b	1548425347456.785
	c	-27.266

D.2.3 135 mm lens

Table D.6 shows the parameters calculated through the regression methods described in Section 6.5.2.3. The hyperbolic least squares model, unlike the 55 mm lens, does not differ significantly from the parameters for the non-linear hyperbolic models. The non-linear hyperbolic regression model without the Z offset assumes an offset of $a = 0m$, while the nonlinear model with the offset calculates that offset to be $a = 0.8m$. The parameters of both of the exponential models are still large due to the location of the Z intercept of the model, but due to the shift of the dataset α values, the intercept is not as large. Note that the two Michaelis-Menten curves have similar parameter values as expected from the similar regression curves.

Table D.6: FFT Method Model Parameters for 135 mm

Model	**Parameter**	**Value**
$\frac{c}{\alpha^b}$ (linear)	b	5.5993
	c	2.1405
$\frac{c}{\alpha^b}$	b	13.505
	c	1.5928
$a + \frac{c}{\alpha^b}$	a	0.64471
	b	20.8624
	c	0.83775
$a + \frac{c}{(\alpha-d)^b}$	a	-0.092995
	b	0.72074
	c	0.30325
	d	0.9089
$\frac{V_{max}\alpha}{K_m+\alpha}$ (linear)	V_{max}	0.15354
	K_m	-0.90191
$\frac{V_{max}\alpha}{K_m+\alpha}$	V_{max}	0.16131
	K_m	-0.89823
$\frac{V_{max}(\alpha-d)}{K_m+(\alpha-d)}$	V_{max}	0.23334
	K_m	-0.58043
	d	0.31959
$be^{c\alpha}$ (linear)	b	254.3071
	c	-4.7509
$a + be^{c\alpha}$	a	0.66411
	b	2975362769.1757
	c	-22.027