University of Alberta

Underwater Stereo Matching and its Calibration

by

Jason Gedge

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Abstract

A fundamental component of stereo vision is that of epipolar geometry. It shows that the corresponding point of a pixel in one image is restricted to a line in another image. When a refractive surface is introduced, such as in underwater imaging, this constraint no longer holds. Instead, the corresponding point of a pixel in one image is now restricted to a curve, not a line, in the other image.

In this thesis, we investigate the impact of a planar refractive interface on stereo matching. We address the issue of 3D point projection in a refractive medium, including cases where the refractive interface is not parallel with the camera's imaging plane. A novel method for calibrating the parameters of a planar refractive interface is proposed. We show how to compute the refractive epipolar curve for a pixel, which allows us to generate a matching cost volume that compensates for the effects of refraction. We implement a multi-view stereo algorithms to test the correctness of our matching cost volume. The experimental results show that our new approach can significantly improve the results of underwater stereo matching over previous approaches using heuristic methods to account for refraction.

Acknowledgements

First, I would like to thank my supervisors Dr. Herbert Yang and Dr. Minglun Gong for all the help and guidance they have provided over the past two and a half years. Through their wisdom I have been able to overcome many obstacles to make it this far. Also, I would like to thank my committee members, Dr. Martin Jagersand and Dr. Dil Joseph, for taking the time to read my thesis and participating in my defense.

My labmates and fellow students – Cheng Lei, Xida Chen, Allen Shen, Matthew Hamilton, and Omar Rodriguez – have also been a great help throughout my studies. Many ideas and helpful tips have been provided through various conversations we have had. I also thank them for taking the time to listen. Additional thanks goes out to Xida Chen for his help with data acquisition in times where I did not have access to the lab.

A big thanks also goes out to Steve Sutphen for all of his help with the camera array and tank in the lab.

I would like to thank NSERC and Alberta Innovates (formerly iCORE and Alberta Ingenuity) for their financial support throughout my studies. With this support, I had the opportunity to attend the top conference in my area of research.

Last, but not least, I would like to thank my family and friends that have always been there, encouraging me and pushing me to do the best I can. I could never have made it this far without their support.

Table of Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Stereo vision is an age-old field in computer vision that concerns itself with the computation of geometry from two or more images of a scene. It is related to the *image correspondence problem*:

> *Given two images $\mathcal{A}, \mathcal{B}$ of a scene and a feature $f \in \mathcal{A}$, find the corresponding feature $f' \in \mathcal{B}$, if it is visible.*

Stereo vision focuses on multiple viewpoints of the same scene, but generally all images are taken at the same time. Once correspondence is established, three-dimensional (3D) points can be obtained by triangulation, allowing us to reconstruct the scene in 3D. A more recent area of interest is focused in researching the impact of underwater environments on stereo vision. Underwater stereo vision has various applications of interest, such as biological observation, robot navigation, and reconstruction of underwater archaeological sites for the purpose of digital archival.

Increasing interest in underwater observation has sparked research in underwater imaging and stereo. NEPTUNE Canada has installed a large seafloor observatory off the west coast of Canada [8]. Their Folger Pass station is equipped with an 8-camera array designed at the University of Alberta and will provide 3D reconstructions of living organisms[1] (Figure 1.1). The Atlantic Innovation Fund (AIF) provides financial support in Atlantic Canada for bleeding-edge innovation. Recently, AIF has provided \$2.2 million of funding for work on various underwater robotics and vision projects, including a vision system that will be used for underwater observation [44]. VENUS is another project providing

[1] The NEPTUNE Canada project has been the major motivation for our own work.

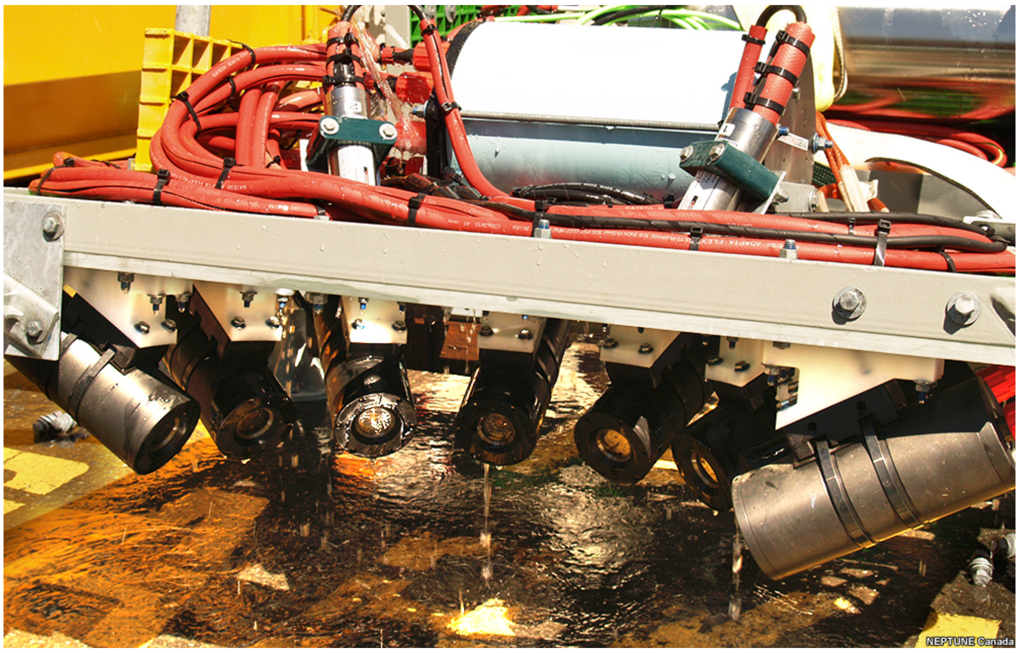


Figure 1.1: The 8-camera array deployed at the Folger Pass station of the NEPTUNE project. This camera array will be used to observe and reconstruct organisms in 3D. (Source: NEPTUNE Canada)

underwater observation data to scientists [48]. Other work has focused on producing large-area photomosaics of the seafloor, often for the purpose of guiding underwater robots [17, 41].

Underwater imaging itself poses many potential challenges. The harsh environment lends itself to turbidity, making it far more difficult to observe features in sufficient detail. Similarly, the effects of light attenuation and scattering are stronger in water, making it very difficult to observe distant objects. Strong currents pose a difficult engineering challenge when designing observational stations that require rigidity and stability. Finally, it is necessary for scenes to be artificially lit at large depth, since sunlight gets absorbed more and more the deeper one travels. All of these challenges make underwater stereo vision difficult, but recent developments in underwater imaging will help pave the way for improved underwater vision [23].

Besides for the challenges in imaging, another major problem exists in underwater stereo vision: *refraction*. Refraction of light occurs when light changes speed, such as when traveling from one medium to another[2]. Cameras will often

[2] Although refraction is often associated with light traveling between two mediums, it can also occur in the same medium when moving between areas of different salinity, temperature, and pressure.

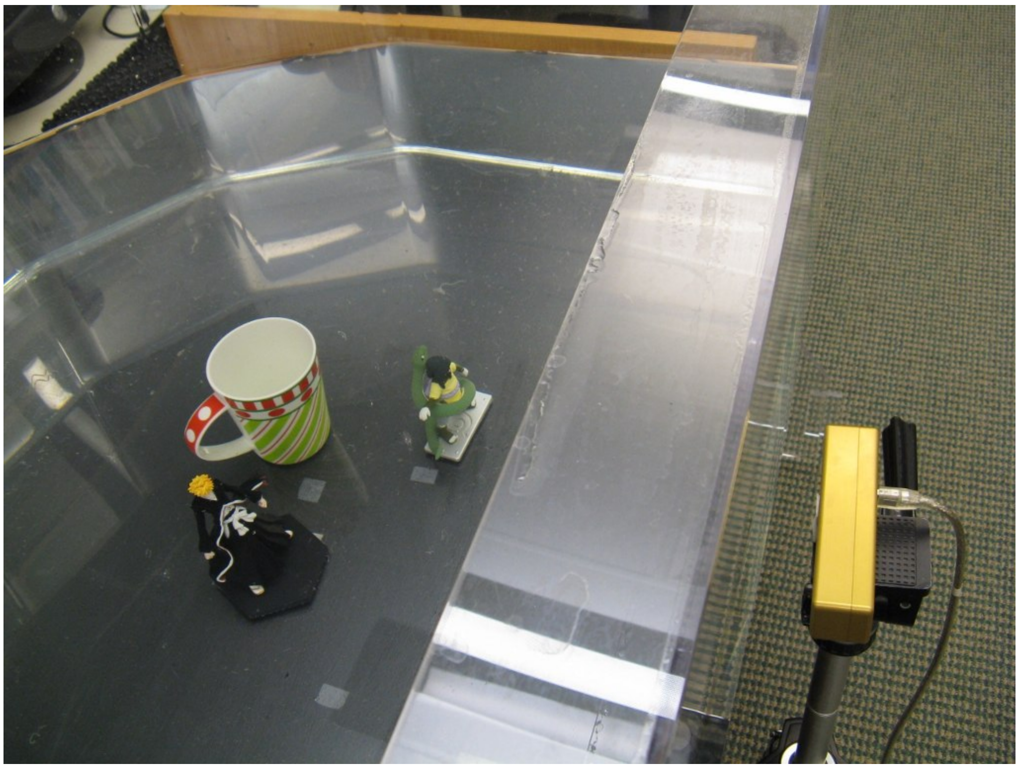


Figure 1.2: Example of a stereo camera observing figures in a water-filled tank.

be placed in waterproof housings with a glass window or outside a tank, which means light has to travel from water to glass, and glass to air before reaching the camera sensor (Figure 1.2). The intrinsic geometry of a calibrated stereo rig, known as *epipolar geometry*, is warped due to the nonlinear nature of refraction. It is important to compensate for this warping to ensure the scene geometry we compute is correct.

Stereo vision algorithms that rely on the underlying epipolar geometry to establish correspondence fall into two major areas: binocular and multi-view. Binocular techniques, as the name suggests, focus on just two views of the same scene. Focusing on only two views naturally results in a computationally easier problem, but less information is available for establishing correspondence. On the other hand, multi-view techniques aggregate the information from three or more views to help with establishing correspondence. Unfortunately more data can also result in a higher chance of making false-positives, so more intelligent outlier detection may be necessary.

In this thesis we present a novel method to address the issue of refraction in stereo. The method consists of two major components:

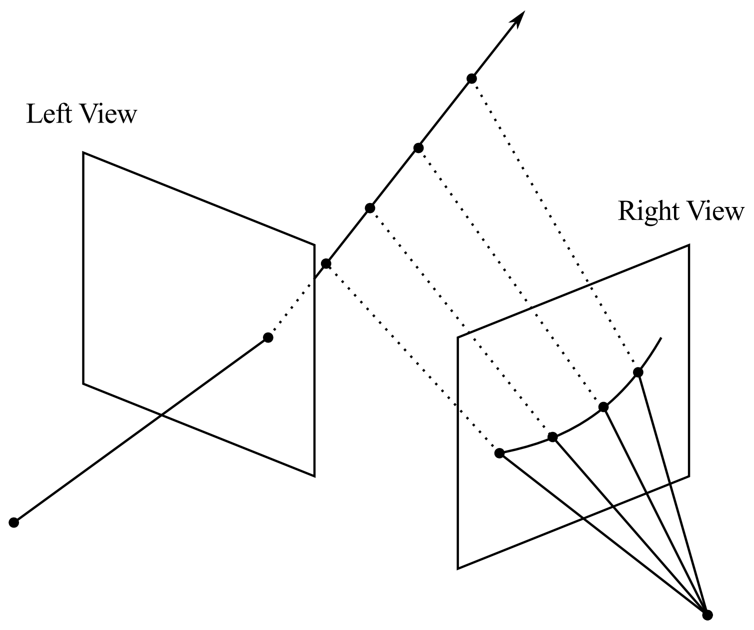
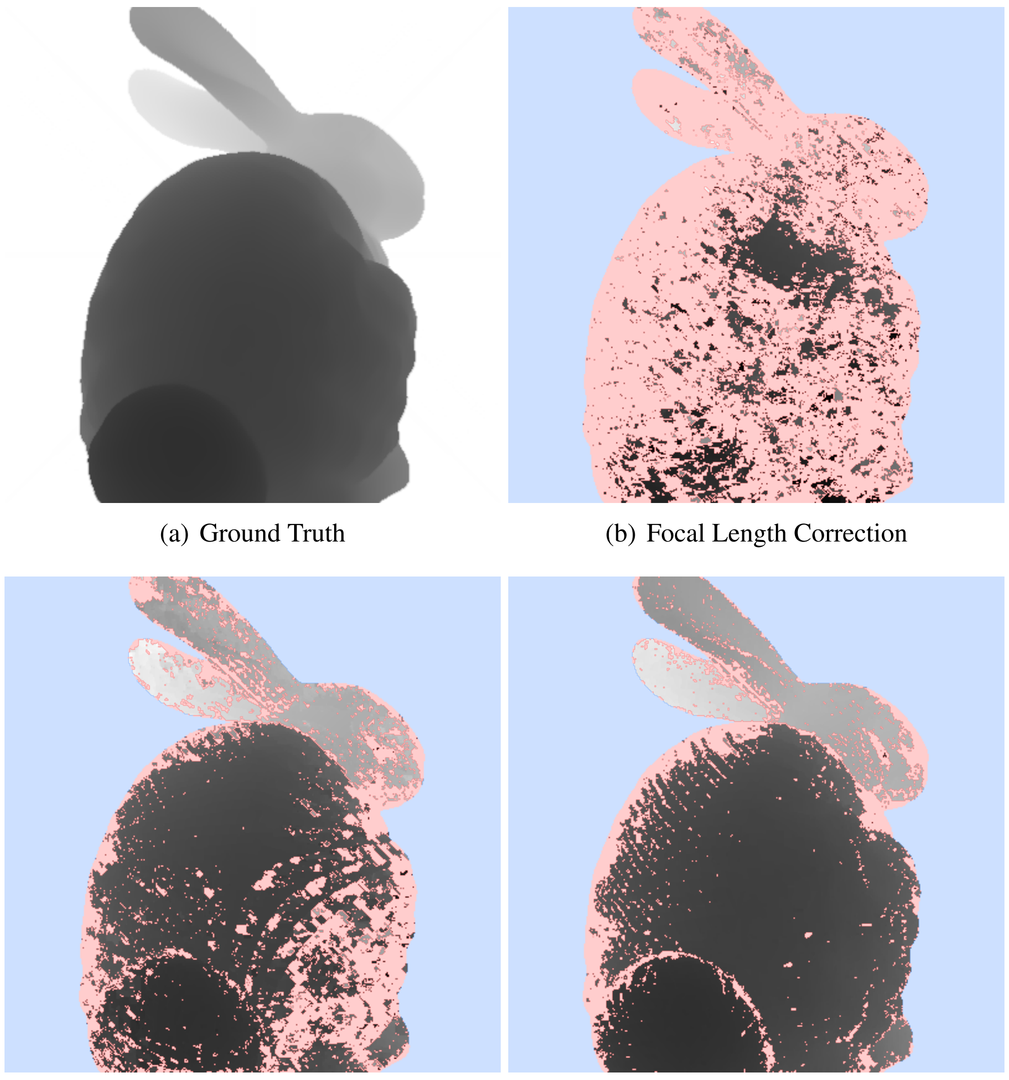


Figure 1.3: Epipolar geometry of two views. A feature in the left view and its corresponding epipolar curve in the right view. In traditional non-refractive epipolar geometry this curve is actually a straight line. The nonlinear effect of refraction warps this line into a curve.

1. a calibration component that uses feature correspondences to find the parameters of each refractive interface, and
2. a way to sample the epipolar geometry associated with a stereo system under the influence of refraction.

We assume a planar refractive interface due to its practicality and mathematical simplicity. Such an interface is common in tanks and aquariums, as well as in camera housings used underwater. Also, as with many stereo algorithms before, we assume a pinhole camera model in all of our derivations. Our calibration component employs a nonlinear optimization to find the refractive interface parameters by minimizing an approximate image-space projection error. Given these parameters, we use a ray-casting technique and refractive projection formula to sample the refractive epipolar geometry imposed by these refractive interfaces. Another major goal of this thesis is to show, using our own system as an example, the importance of using a physical model for refraction when performing stereo matching.

Existing underwater stereo matching techniques approach refraction in one of three ways:



(a) Ground Truth (b) Focal Length Correction

(c) Radial Distortion Correction (d) Our result

Figure 1.4: A comparison of depth maps produced from our approach and two traditional approaches (focal length correction and radial distortion correction). Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink. Our result provides a dense, smooth depth map. Using a naive focal length correction method which ignores refraction results in many maching failures.

1. ignoring refraction completely,
2. using approximations as a heuristic to guide search, and/or
3. modeling refraction as radial lens distortion.

Treibitz et al. show that all of these assumptions can be very erroneous and a physically accurate model is necessary for correcting refraction [46]. Approximate methods usually note how the focal length of a camera is scaled underwater [12], yet these approximations are at a very high-level and still do not modify the epipolar geometry correctly to compenstate for refraction. Two-dimensional search is still necessary to help find correspondences [12]. Radial lens distortion has also been used (e.g. [40]) to model refraction. These methods commonly "absorb" refractive distortions into the radial lens distortion parameters of the camera during calibration. Unfortunately, as we will show in Chapter 2, the projection of a point depends on its 3D location. Since lens distortion only compensates for image-space distortions, it is insufficient for refraction correction.

Our method projects multiple samples of a refracted ray into the image to obtain the refractive epipolar geometry. To ensure physical correctness, point projection is also modeled with refraction. This allows us to obtain an accurate representation of the refractive epipolar curve (Figure 1.3). One of the advantages of this approach is that no refraction-specific algorithms need to be developed; our refractive epipolar curve simply replaces the traditional epipolar geometry used in any existing method. To showcase this advantage, we incorporate our refractive epipolar geometry into the method of Campbell et al. [7] (Figure 1.4).

The remaining part of this thesis is organized as follows. Previous work in binocular/multi-view stereo and work related to underwater stereo vision is reviewed in Chapter 2. Chapter 3 introduces the theory of refraction and derives the various expressions necessary for estimating the refractive epipolar geometry. Interface calibration and underwater stereo matching are presented in Chapter 4, followed by various experiments in Chapter 5. Finally, we conclude our work and suggest possible avenues of future work in Chapter 6.

Chapter 2

Related Work

Stereo vision covers a large area of interest in computer vision, and much research has been dedicated to improving results in both binocular and multi-view stereo. This has been driven by the availability of existing datasets with ground truths [37,39]. These datasets allow researchers to evaluate the accuracy and robustness of their techniques and to better compare their techniques against those of other researchers.

Underwater stereo vision is an area that has been studied far less, with a large portion of the research falling within the past decade. Various approaches are taken, but most of them use some form of a high-level approximation, based on heuristics, which does not properly take into account the refractive epipolar geometry. In this chapter we introduce recent advances in binocular stereo and multi-view stereo, followed by related work in the area of underwater stereo matching.

2.1 Binocular Stereo

A method that concerns itself with reconstructing depths from *exactly* two views is called *binocular stereo*. Binocular stereo methods commonly take one of two approaches: *local* and *global*. In this section we begin by describing some fundamental terminology for binocular stereo, which is followed by discussion of the distinguishing features between local/global methods and various state-of-the-art techniques.

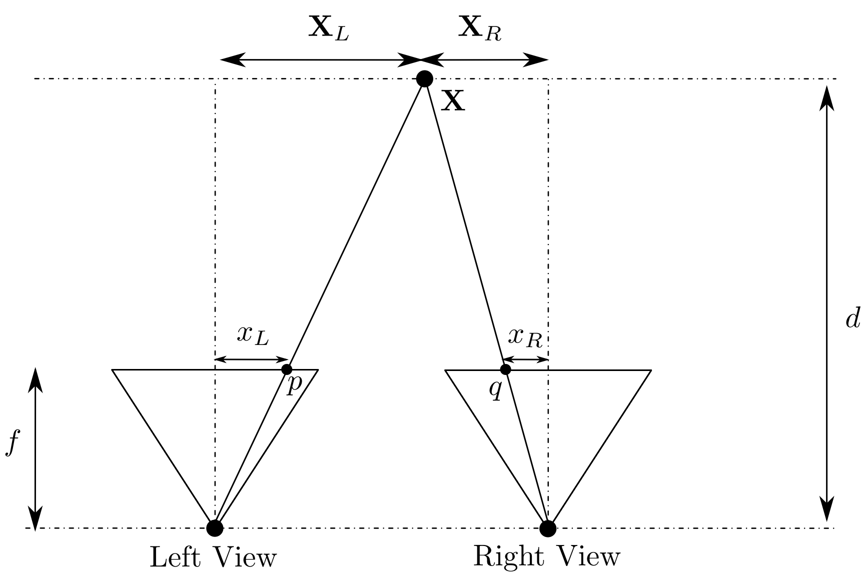


Figure 2.1: Binocular stereo setup. A point $\mathbf{X}$ is projected to pixels p, q in two views having a focal length of f. $\mathbf{X}$ has a depth of d with respect to the two views.

2.1.1 Terminology

Figure 2.1 shows a typical binocular stereo setup with two cameras having parallel viewing directions. In particular, once lens distortion is removed and imaging planes become coplanar (with the same orientation about the optical axis) we have a set of *rectified* views[1]. Rectified views make the search for correspondence simpler because corresponding points will lie in the same row of both images.

Two important concepts in binocular stereo are *depth* and *disparity*[2]. The depth of a point $\mathbf{X}$ is the perpendicular distance from the views's center-of-projection to $\mathbf{X}$, i.e., d in Figure 2.1. The disparity of a point $\mathbf{X}$ is the shift in pixel location between two views, i.e., $x_L - x_R$ in Figure 2.1[3].

Now consider the pixel locations x_L and x_R. We can relate $\mathbf{X}$ to these locations by using similar triangle ratios:

$$x_L = \frac{f\mathbf{X}_L}{d}, \; x_R = \frac{f\mathbf{X}_R}{d}, \tag{2.1}$$

[1] If views are not initially rectified, methods exist to perform rectification (e.g., [14]).

[2] The traditional notion of disparity does not apply in the situation where there is a refractive interface, so we choose to use the more general concept of depth.

[3] Consideration should be taken with regards to sign. For example, in Figure 2.1 $\mathbf{X}_R$ is negative, since it is to the left of the right view's viewing direction.

where $\mathbf{X}_L$ is the location of $\mathbf{X}$ from the left view's frame of reference, and similarly $\mathbf{X}_R$ is the location of $\mathbf{X}$ in the right view's frame of reference. From (2.1) we get:

$$x_L - x_R = f\frac{\mathbf{X}_L - \mathbf{X}_R}{d}. \tag{2.2}$$

As one can see from (2.2), if we increase the depth then the disparity will decrease, and vice versa. Hence, depth and disparity are inversely proportional to each other.

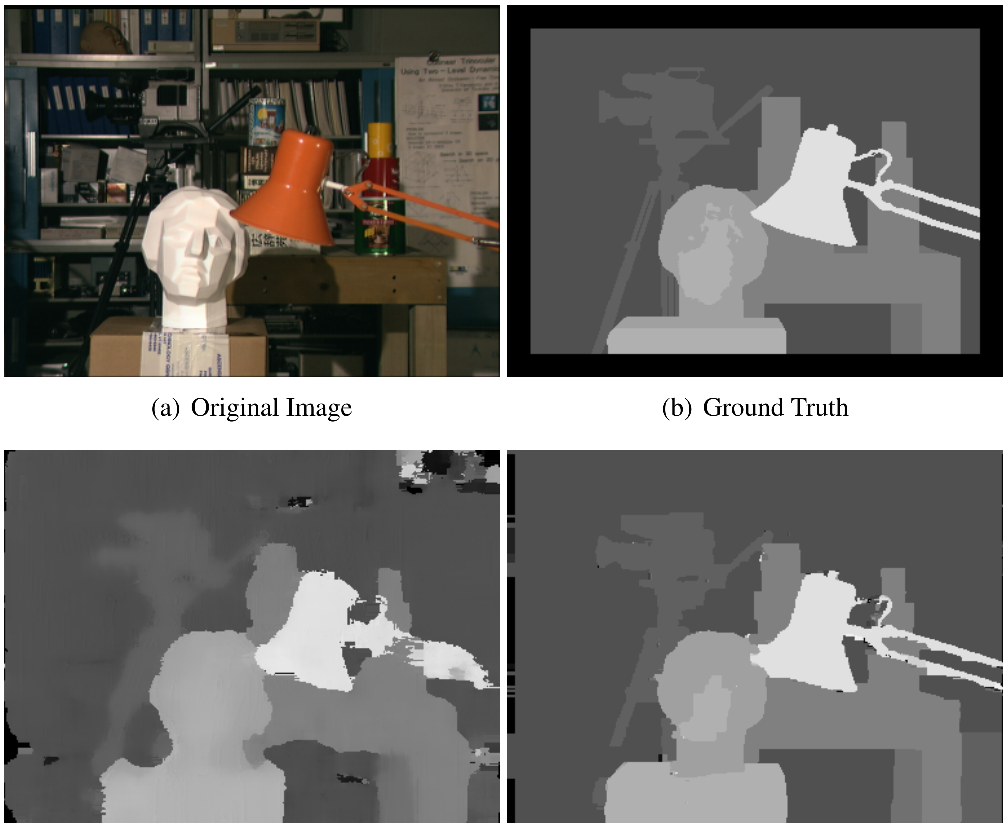
2.1.2 Local Methods

Local techniques consider each pixel in isolation of all others. Computed depths are not compared to neighbouring depths to enforce any continuity constraints or smoothness. As a result, the depth/disparity maps computed using local methods are often noisier than those computed using global techniques (Figure 2.2). Local techniques often employ a winner-take-all approach: for pixel p, given a set a depth hypotheses, D, and a corresponding cost function, c_p, we select the depth hypothesis with the lowest cost:

$$\arg\min_{d \in D} c_p(d). \tag{2.3}$$

One of the major distinguishing factors of local methods is the cost function c_p. Given a pixel $p = \{p_x, p_y\}$ in one view, a given depth hypothesis d will correspond to some pixel p' in the other view. A common technique for computing $c_p(d)$ is to compare the similarity of square windows centered at p and p' [20]. This is just one possible approach to computing cost, but other techniques exist, such as filtering and probabilistic measures [20].

With window-based techniques a trade-off has to be made with respect to the window size. Smaller windows are more efficient and more likely to cover the same surface, but may not contain enough information to disambiguate true positive matches from false positive matches. Larger windows alleviate this problem, but are more likely to extend beyond occlusion boundaries and other depth discontinuities. State-of-the-art techniques provide excellent results. In particular, segmentation-based schemes have proven to be effective [21, 55]. Segmentation is the process of grouping together similarly colored regions of pixels. A typical assumption is that small regions of similar color share similar depths, known as the gestalt principal of organization by similarity [55].


(a) Original Image (b) Ground Truth

(c) Local Technique [4] (d) Global Technique [25]

Figure 2.2: A comparison of a local and global binocular stereo techniques. Disparity maps computed by local methods are more susceptible to noise since the disparities of neighbours are not considered. Global methods use neighbouring depth estimates to help "smooth" the result. Brighter pixels correspond to regions of high disparity (i.e., small depth), ranging from the closest objects (white) to the farthest (dark grey).

One approach to similarity grouping is to weight pixels in a window based on their similarity to the color of the center pixel in the window. Yoon and Kweon propose a weighting scheme they name *adaptive support weights* [55], which is similar to the technique of bilateral filtering [45]. For every pixel p we first compute a spatial Gaussian:

$$w_{\text{spatial}}(p;q) = \exp\left(-\frac{\Delta d_{pq}^2}{\gamma_{\text{spatial}}}\right), \tag{2.4}$$

where q is the center pixel in a window, Δd_{pq} the Euclidean distance between the location of pixels p and q, and γ_{spatial} a user-defined rate-of-decay parameter. Also, a chromatic Gaussian is computed:

$$w_{\text{chromatic}}(p;q) = \exp\left(-\frac{\Delta c_{pq}^2}{\gamma_{\text{chromatic}}}\right), \tag{2.5}$$

where Δc_{pq} is the color difference between pixels p and q, and $\gamma_{\text{chromatic}}$ a user-defined rate-of-decay parameter. The final weight of pixel p is then the product of these two, i.e.,

$$\begin{aligned} w(p;q) &= w_{\text{spatial}}(p;q) \cdot w_{\text{chromatic}}(p;q) \\ &= \exp\left(-\frac{\Delta d_{pq}^2}{\gamma_{\text{spatial}}}\right) \exp\left(-\frac{\Delta c_{pq}^2}{\gamma_{\text{chromatic}}}\right). \\ &= \exp\left[-\left(\frac{\Delta g_{pq}^2}{\gamma_{\text{spatial}}} + \frac{\Delta c_{pq}^2}{\gamma_{\text{chromatic}}}\right)\right] \end{aligned} \tag{2.6}$$

Hosni et al. provide another approach to similarity grouping that results in a robust local segmentation [21]. The cost $d(p,q)$ of moving from pixel p to a neighbouring pixel q is defined as the Euclidean distance between their respective color values:

$$d(p,q) = \sqrt{(p_r - q_r)^2 + (p_g - q_g)^2 + (p_b - q_b)^2}, \tag{2.7}$$

where p_r, p_g, and p_b are the red, green, and blue values of pixel p, respectively.

Based on this distance metric we can develop the notion of shortest path between two pixels. Given a window centered at pixel p, let $S(q)$ be the length of the shortest path from p to q given the distance metric in (2.7). Hosni et al. specify the weight of pixel q to be:

$$w(q) = \exp\left(-\frac{S(q)}{\gamma}\right), \tag{2.8}$$

where γ is a user-defined parameter to adjust the "sharpness" of the segmentation.

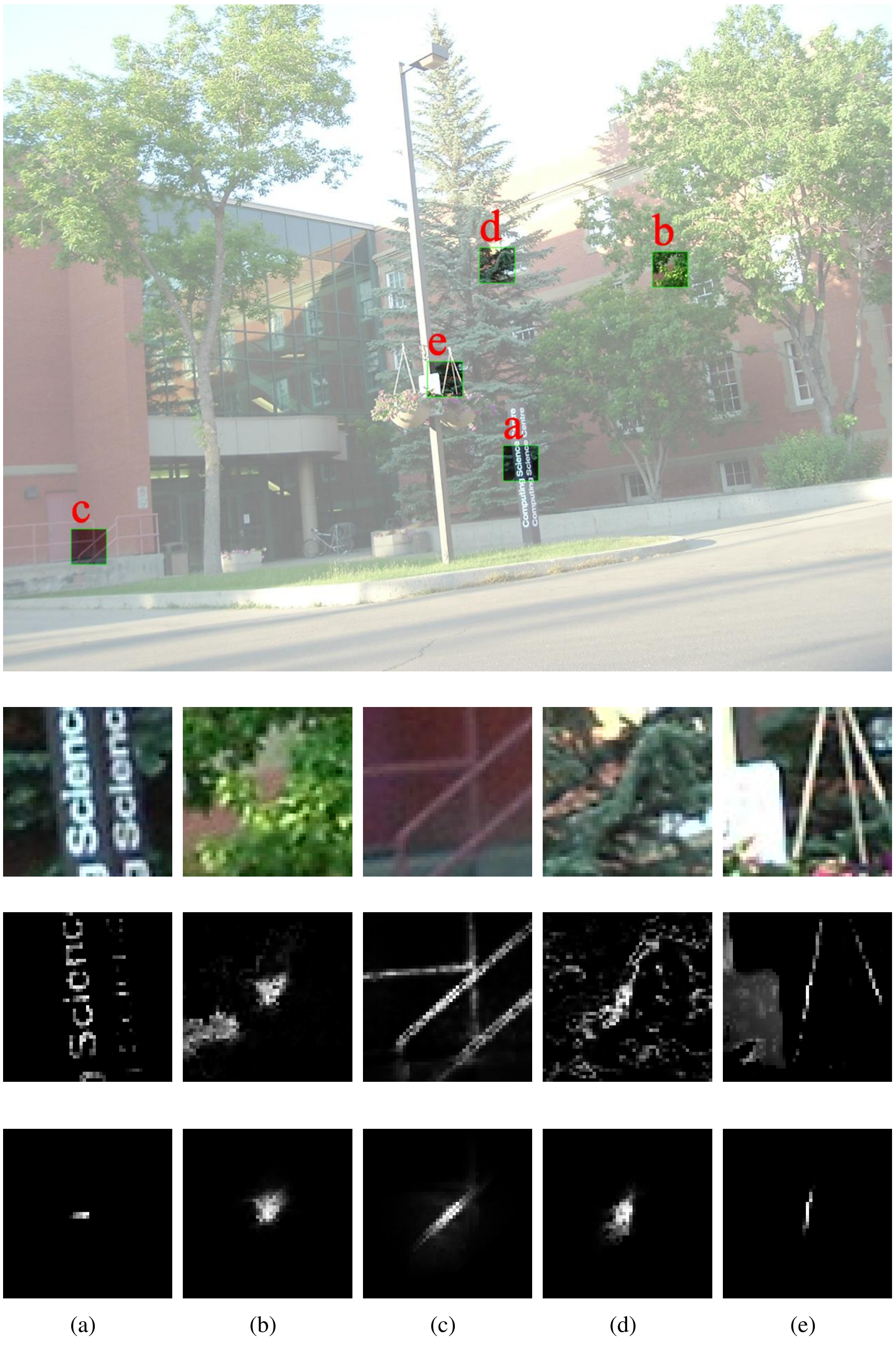
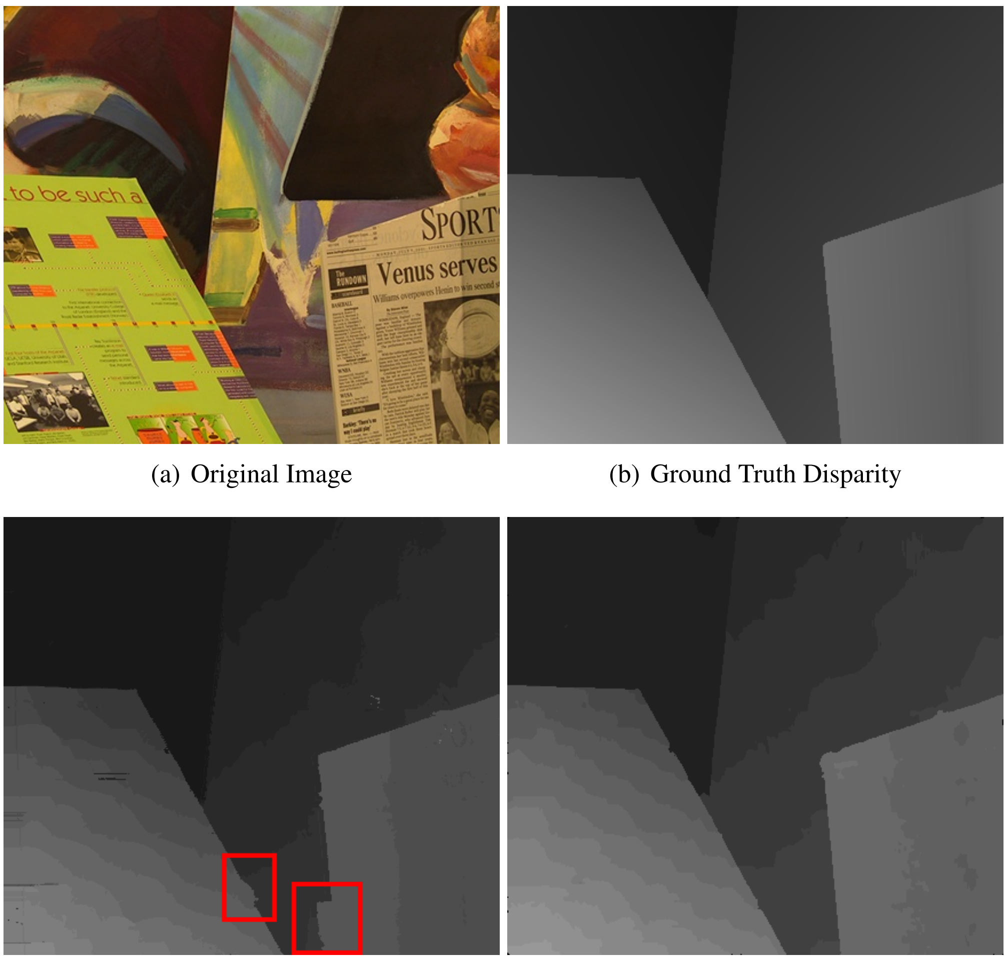


Figure 2.3: From top to bottom: five windows are taken from an image, zoomed in versions of these windows, pixel weights with adaptive weights [55], and pixel weights with geodesic support weights [21]. Brighter pixels indicate a larger weight. Both of these weighting schemes seek to weight regions of similar color higher, with respect to the center pixel, but the method of Hosni et al. provides a harder cutoff.



(a) Original Image (b) Ground Truth Disparity

(c) Adaptive Weights [55] (d) Geodesic Support Weights [21]

Figure 2.4: Comparison of adaptive and geodesic support weights in a local stereo method. The harder segmentation of geodesic support weights provides better results, especially in areas with sharp depth discontinuities (highlighted regions in (c)). Brighter pixels correspond to regions of high disparity (i.e., small depth), ranging from the closest objects (white) to the farthest (black).

Figure 2.3 visualizes adaptive and geodesic support weights for various windows within an image. Adaptive support weights [55] often give higher weights to background pixels, which are likely to have a true depth different from that of the center pixel. On the other hand, geodesic support weights result in a harder segmentation which is more likely to give very low weights to background pixels.

2.1.3 Global Methods

Global methods consider not only a cost for each pixel, but also the chosen depths of neighbouring pixels. By enforcing constraints on how much neighbouring depths can differ, one can "smooth" the computed depths (Figure 2.2). State-of-the-art global methods are often formulated as Markov Random Field (MRF) optimization problems [43]:

$$E = \sum_{p} c_p(l_p) + \lambda \sum_{\{p,q\}\in\mathcal{N}} V_{pq}(l_p, l_q). \tag{2.9}$$

l_p represents a label corresponding to a pixel p. In the case of stereo matching, a label maps to a depth hypothesis. The first term in the above sum is called the *data term*, which is simply the cost function used in local methods. The second term is called the *smoothness term*. $V_{pq}(l_p, l_q)$ is a smoothness function that returns a cost associated with assigning labels l_p and l_q to pixels p and q, respectively. λ is a parameter to specify how much influence the smoothness term has on the total energy. If $\lambda = 0$ then (2.9) reduces to the same problem as (2.3). A recent survey of various methods to solve these optimization problems was presented by Szeliski et al. [43].

Finding the global minimum of (2.9) is known to be NP-hard problem when there are more than two labels [5]. This implies that unless P = NP, finding the global minimum is computationally intractable in the general case. Nevertheless, many algorithms exist that efficiently find local minimums for (2.9). Two well-known optimization algorithms are *graph cuts* and *belief propagation*.

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, w : \mathcal{E} \mapsto \mathbb{R}\}$ be a graph with edges weighted by w, and let $\{s, t\} \subset \mathcal{V}$ be the *source* and *terminal* vertices. A cut $\mathcal{C} = \{\mathcal{V}^s, \mathcal{V}^t\}$ is then defined as a partition of $\mathcal{V}$ (i.e., $\mathcal{V}^s \cap \mathcal{V}^t = \emptyset$) such that $s \in \mathcal{V}^s$ and $t \in \mathcal{V}^t$ [25] (Figure 2.5). Intuitively, we can see this as selecting a set of edges which separates the source and terminal verticies when removed. The *cost* of $\mathcal{C}$ is defined as:

$$|\mathcal{C}| = \sum_{e\in\mathcal{E}} \begin{cases} w(e) & \text{if } e \cap \mathcal{V}^s \neq \emptyset \text{ and } e \cap \mathcal{V}^t \neq \emptyset \\ 0 & \text{otherwise} \end{cases}. \tag{2.10}$$

A *min-cut* is simply a cut with minimal cost over all possible cuts. Boykov et al. describe an algorithm that uses graph cuts to optimize (2.9) [5].

Belief propagation (BP) uses a message-passing paradigm for minimizing (2.9), where message vectors are passed between adjacent pixels over a series of iterations [11]. Max-product belief propagation focuses on maximizing the probability that a pixel should have a specified label. To more easily fit into

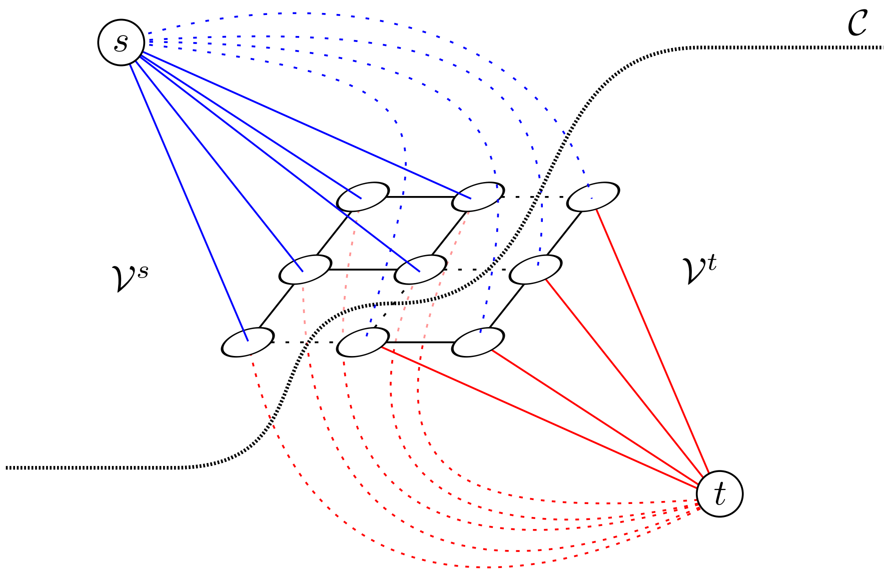


Figure 2.5: Example of a graph cut. The cut $\mathcal{C}$ partitions the graph into two groups: $\mathcal{V}^s$, all vertices attached to the source s, and $\mathcal{V}^t$, all vertices attached to the terminal t. The dashed lines correspond to those edges that were "cut".

(2.9), the max-product formulation is often transformed into a negative log-likelihood formulation, which in turn results in a minimum sum problem.

For (2.9), the message vectors are simply K-dimensional vectors where K is equal to the total number of labels. Let m_{pq}^t be a message passed from pixel p to neighbouring pixel q at iteration t. The message vectors are initialized to zero, i.e., $m_{pq}^0 = \mathbf{0}$. For iterations $t = 1 \dots T$ the message vectors are:

$$m_{pq}^t(l_q) = \min_{l_p} \left(V_{pq}(l_p, l_q) + c_p(l_p) + \sum_{s \in \mathcal{N}(p) - \{q\}} m_{sp}^{t-1}(l_p) \right), \tag{2.11}$$

where $\mathcal{N}(p)$ is the set of pixels that are neighbours of p. As Weiss describes, the message $m_{pq}(l_q)$ can be regarded as pixel p telling pixel q how it fits label l_q given the local information p has and q does not [52]. In particular, the local information that only p has is its cost function c_p and the messages sent to it from its neighbours other than q (i.e., $\mathcal{N}(p) - \{q\}$).

After T iterations, belief vectors b_p are computed as:

$$b_p(l_p) = c_p(l_p) + \sum_{q \in \mathcal{N}(p)} m_{qp}^T(l_p). \tag{2.12}$$

The term *belief vector* stems from the max-product formulation of BP. In such a formulation we are dealing with probabilities, so $b_p(l_p)$ represents the probability (i.e., belief) that p has label l_p. Since we are working with negative log-likelihoods, the probability is maximized when the sum is minimized. Hence, the label l_p^* selected for pixel p is the label which has the smallest belief value:

$$l_p^* = \arg\min_{l_p} b_p(l_p). \tag{2.13}$$

2.1.4 Cross-checking

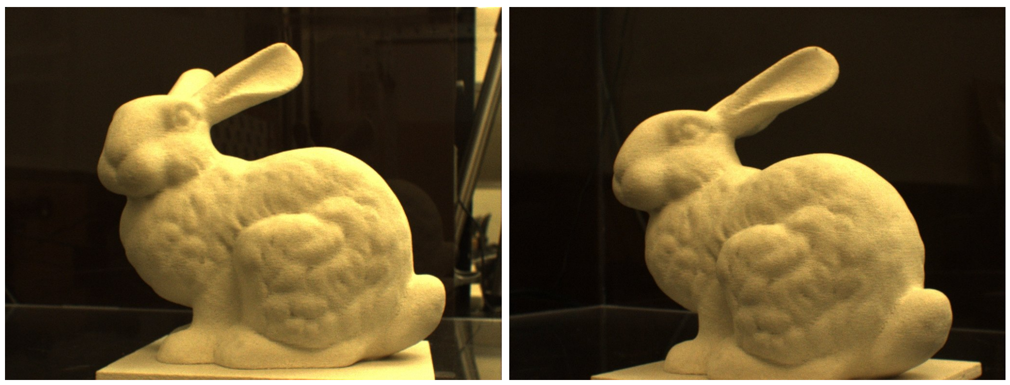
Other techniques can be used in tandem with local/global methods. One particularly important technique is *cross-checking*, which is essentially a consistency check between the left and right views to ensure that depth hypotheses in one view correspond to similar depth hypotheses in the other view. Given a depth hypothesis d in one image, we can obtain a 3D point $\mathbf{x}$. Now if we project $\mathbf{x}$ into the other view, we should get another depth hypothesis d'. Similarly, from this hypothesis we can obtain another 3D point, $\mathbf{x}'$. To ensure these depth hypotheses are consistent, the two points should be close together. In other words, the points should satisfy:

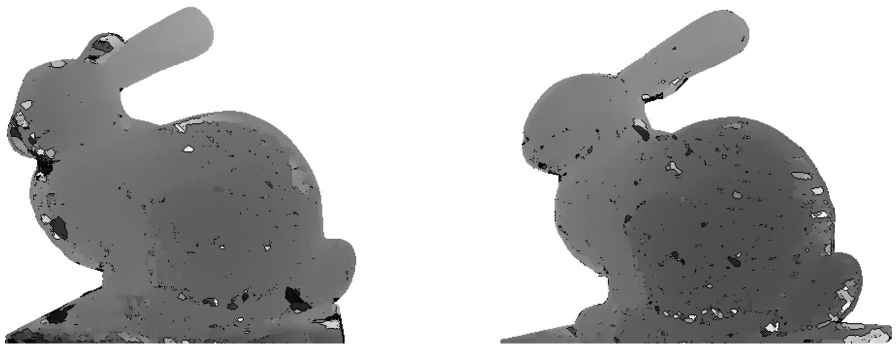
$$\| \mathbf{x} - \mathbf{x}' \| \leq \alpha, \tag{2.14}$$

where α is a user-specified constraint on the maximum distance between the two depth hypotheses. If this check fails then we discard the depth hypothesis d due to inconsistency. Note that the passing of this check does not offer any correctness guarantees for our depth hypotheses, just that $\mathbf{x}$ and $\mathbf{x}'$ are consistent between views. Figure 2.6 shows the result of applying cross-checking to a pair of depth maps. Since only three views were used, parts of the bunny were occluded in some views. These occluded areas provided false positive matches which were inconsistent across views, and hence were removed by the cross-checking phase.

2.2 Multi-view Stereo

An area that is gaining recent popularity is that of multi-view stereo. A focus of multi-view stereo is to reconstruct 3D models of objects from a set of calibrated views [39]. As Seitz et al. describe in their survey, multi-view techniques are incredibly diverse, with a large set of characteristics. Some important characteristics include:


(a) Original images


(b) Before cross-checking

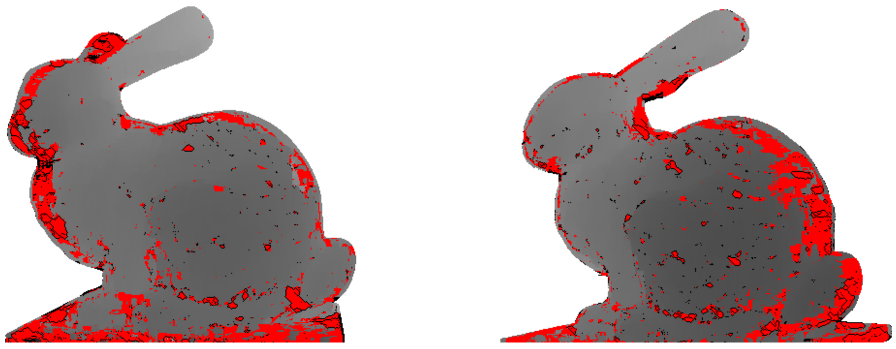
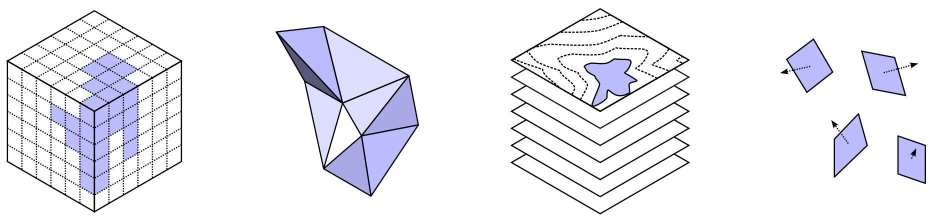

(c) After cross-checking

Figure 2.6: Comparison of depth maps before and after cross-checking. Cross-checking removes inconsistent hypotheses (red pixels), which exist due to matching failure. Note, white pixels outside of the silhouette of the bunny have no depth hypothesis.



(a) Voxel Grid (b) Triangular Mesh (c) Level Sets (d) Surfels

Figure 2.7: Four scene representations used in multi-view stereo reconstruction.

- *Scene representation*: How is the result represented? Triangular mesh? Surfels? Voxels?
- *Photo-consistency measure*: Given, for example, a point, how can we measure its consistency across multiple views? In other words, how can we determine from our images whether or not that point truly does belong to a surface in the scene.
- *Visibility model*: How can we establish whether a points is visible in a view, or whether it is occluded?
- *Reconstruction algorithm*: Given all of the above characteristics, how can we put them all together to reconstruct a scene? Should we evolve a surface from an initial estimate? Should we establish a sparse set of points followed by expanding and filtering this set?

2.2.1 Scene Representation

The choice of representation for scene geometry is a matter of balancing various pros and cons. A voxel occupancy grid (Figure 2.7(a)) splits the visible scene into cells where each cell is labelled as occupied or vacant [49]. One of the major benefits of a voxel occupancy grid is its simplicity. Another benefit is the built-in neighbouring relationship imposed by the grid structure. Unfortunately, the quality of the result is highly dependent on the resolution of the grid. Considering this, a voxel occupancy grid is not applicable to large scenes.

A triangular mesh (Figure 2.7(b)) is a set of connected triangles, often forming a closed surface [27, 50]. Although triangular meshes are one of the more complicated representations to incorporate algorithmically, there are many advantages to using triangular meshes. In particular, since they directly represent surfaces, visibility assessment is straightforward. Also, modern graphics processing units use triangles as the basic rendering unit. Finally, meshes can have a fine or coarse resolution as necessary to accommodate for the complexity of the scene.

Level sets (Figure 2.7(c)) tessellate space in one dimension, and for each two-dimensional slice of this tessellation a *signed distance map* is stored [34]. That is, for each point in the signed distance map, a value is stored which gives its distance from an underlying surface. This is a very powerful and robust representation, but has the downside of needing a large amount of memory. This makes them less appropriate for high-resolution reconstruction. In spite of that, since signed distances are stored, more intelligent reconstruction can be performed than voxel occupancy grids.

Surface elements, otherwise known as surfels (Figure 2.7(d)), are simply oriented patches or discs [13]. The sparsity of such a representation makes it a robust representation that can handle any scene geometry effectively. The downside is that if a set of surfels is too sparse, holes may exist in reconstructed models. Depth maps can also be used in a similar fashion to surfels [6].

2.2.2 Photo-consistency Measure

Photo-consistency measures used in multi-view stereo are very much related to those used in binocular stereo (e.g., c_p in (2.3)). The major difference is that there are a larger set of views to compare, so aggregation of these costs is of key importance.

One other benefit of multi-view algorithms is that we often have an estimate of scene geometry. This estimate can be used to select better samples for comparing photo-consistency [13]. Traditional binocular stereo algorithms often use square windows in both views, but a square window in one view corresponds to points that rarely project to a square window in another view (Figure 2.8). If we take samples from estimated geometry and project these samples back into images, we can more effectively measure the photo-consistency. This approach is applicable only to algorithms that iteratively update the estimated geometry.

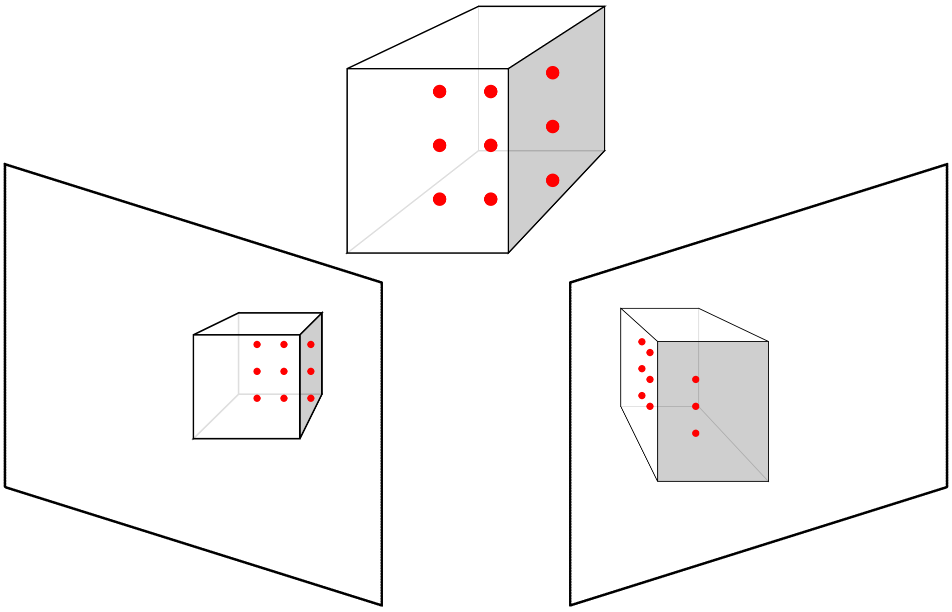


Figure 2.8: Corresponding samples (e.g., for measuring photo-consistency) to select in the right image given the square window of samples in the left image. The right side of the cube has been shaded in all views. Traditional binocular stereo methods often use a square windows in both views, yet samples of a square window in one view are often warped in another.

2.2.3 Visibility Model

Visibility models provide heuristics to multi-view algorithms so as to reduce the number of images required for photo-consistency measurement, and also to help filter geometry estimates. As mentioned previously, some scene representations allow one to explicitly assess the visibility of a point (e.g., triangular meshes [50]). Other approaches are more heuristic, such as selecting views that are nearby, using intelligent outlier detection, or even a combination of the two [7].

2.2.4 Reconstruction Algorithm

The reconstruction algorithm, as its name suggestes, is the part of a multi-view algorithm that reconstructs scene geometry. One approach is to formulate the problem as a cost function over a 3D cost volume, and then optimize this cost function [27, 49]. This is quite common when representing scene geometry using a voxel occupancy grid. One can also take an initial surface estimate and, over various iterations, evolve this surface so as to minimize some cost

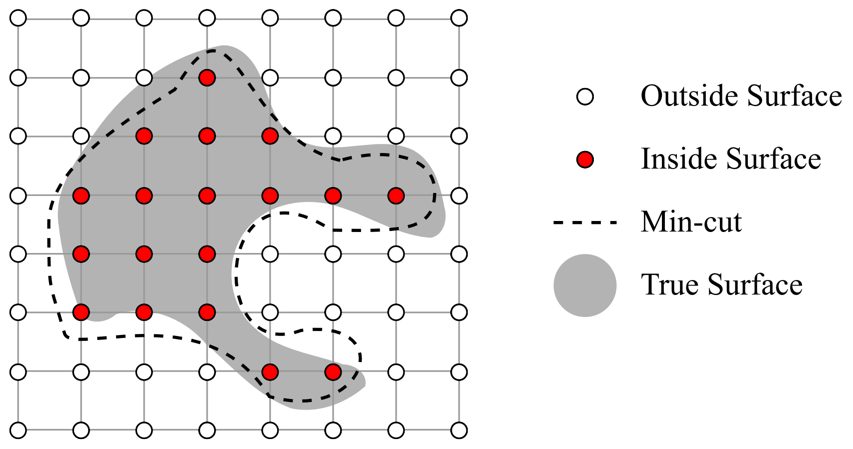


Figure 2.9: A visualization of the method of volumetric graph cuts. A two-dimensional slice is shown with the graph nodes for the slice. The cut removes edges that cross the surface, i.e., separates the graph into subgraphs that reflect areas "inside" and "outside" the surface.

function [50]. These methods generally operate on voxel occupancy grids, level sets, and meshes [39]. One can also compute a set of depth maps, enforcing consistency across multiple views, instead of just two as is the case in traditional binocular stereo [6]. Finally, some methods focus on extracting features and associating 3D points with these features [13].

2.2.5 Methods

Vogiatzis et al. introduce a method referred to as *volumetric graph cuts*. At a high level, they construct a three-dimensional grid graph with a special weighting on the edges such that edges that cross a true surface have low weight. A min-cut (Section 2.1.3) on this volume can then be intuitively viewed as "carving" out the surface from this grid graph (Figure 2.9). This simple high-level explanation makes volumetric graph cuts an excellent method for introducing multi-view reconstruction.

The most important part of their method is supplying weights for the edges. To ensure a high quality cut it is important to have low weights exist only near the surface. To ensure this Vogiatzis et al. introduce a voting-based weighting

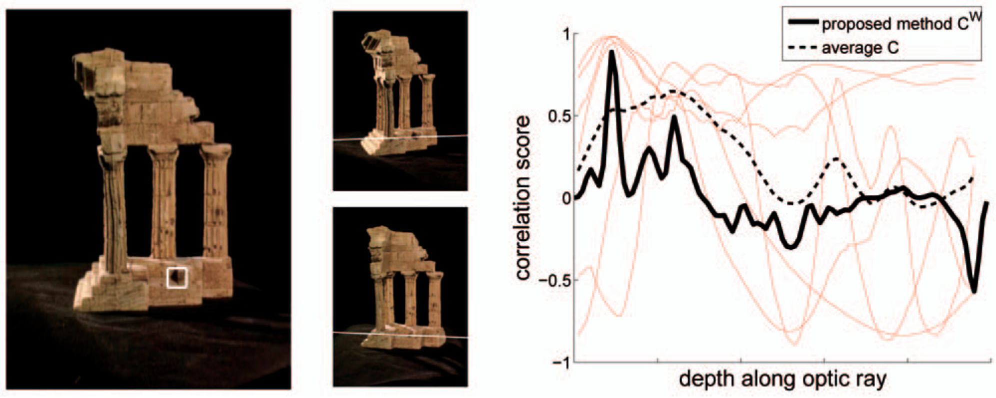


Figure 2.10: Aggregated correlation curves in the method of Vogiatzis et al. [49]. Simply averaging the curves does not take into account depth discontinuities, and hence the peak of the average curve does not correspond to the true depth. The sliding Parzen window technique used by Vogiatzis et al. obtains a peak at the correct depth. (Source: [49])

scheme. They compute the photo-consistency measure $\rho(\mathbf{x})$ of a 3D point $\mathbf{x}$ as:

$$\rho(\mathbf{x}) = \exp\left(-\mu \sum_{i=1}^{N} \mathrm{VOTE}_i(\mathbf{x})\right), \tag{2.15}$$

where N is the number of images; μ, a user-defined parameter; and $\mathrm{VOTE}_i(\mathbf{x})$, the vote offered by image i for $\mathbf{x}$. The vote is calculated in three steps. First, the projection of $\mathbf{x}$ into image i is used to find epipolar lines in neighbouring views. Normalized cross-correlation (NCC) curves are then computed along these lines and aggregated using a sliding Parzen window (Figure 2.10). If the peak of this aggreated curve occurs at a depth corresponding to $\mathbf{x}$, then VOTE_i is set to the NCC correlation score at that point. Otherwise, VOTE_i is set to zero. After setting these weights to the edges in the 3D grid graph, a mininmum graph cut is obtained to find the voxels that correspond to the surface.

One of the leading algorithms (with regards to the Middlebury evaluation [39]) is that of Furukawa and Ponce [13]. They emphasize several strengths of their algorithm:

- there are no initialization requirements,
- a sparse scene representation that is robust, and
- intelligent outlier detection.

They represent the scene using a set of oriented rectangular patches (Figure 2.11). In particular, a patch p is represented by:

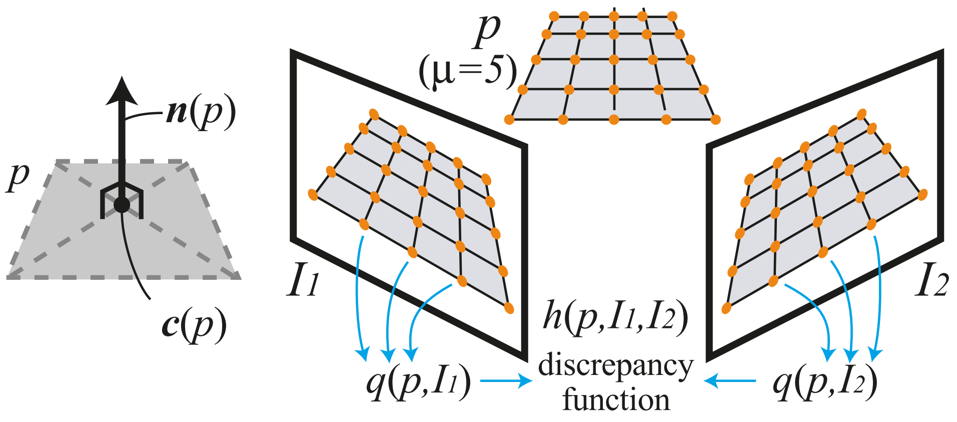


Figure 2.11: The patch representation used by Furukawa and Ponce [13] (Left). A uniform sampling grid on the patch is projected into the image (Right). (Source: [13])

- $\mathbf{c}_p$, the center of patch p
- $\mathbf{n}_p$, the normal of patch p, and
- R_p, the reference image for patch p.

Their algorithm works in three phases: initialization, expansion, and filtering. In the initialization phase, various features are detected in all images, and correspondence is established between these features. Given a correspondence, a patch candidate is estimated. The photo-consistency of this candidate is then tested, and if it is above some threshold then the candidate is stored for the expansion and filtering stages.

The expansion phase, as its name suggests, attempts to expand the current set of candidate patches so that there is at least one patch per cell in all of the images. A cell is a $\beta \times \beta$ square of pixels in the image, where β is a user-defined parameter. Hence, the smaller the cell size, the denser the reconstructed set of patches. Empty cells are then initialized with patch candidates from neighboring cells. The parameters of these candidates are then optimized so as to reorient and relocate the candidates to fit the unknown geometry viewed within that cell.

The last phase filters this expanded set to remove outliers. The filtering is based on several visibility heuristics. The expansion and filtering phase can be repeated several times to remove erroneous patches. Figure 2.12 shows an example of a reconstruction of the Stanford bunny from eight images.

In this thesis, we implement the method of Campbell et al. [7]. Campbell et al. observe that an NCC peak along an epipolar line does not necessarily indicate a correct correspondence. In particular, they focus on compensating for two major issues:

1. repetitions in texture (leads to false positives), and
2. occlusion, distortion, and lack of texture (complete matching failure).

For the first issue, a spatial consistency constraint is imposed to ensure that selecting an NCC peak in one pixel results in a depth hypothesis that is similar to the chosen depth hypotheses of its neighbours. For the second issue, the authors propose an "unknown" label into the optimization which identifies pixels as having a depth that cannot be easily established.

Campbell et al. formulate their problem as an MRF (Section 2.1.3). In particular, for each pixel p in a reference image, the top K NCC peaks are obtained from the epipolar lines in N neighbouring views. For each of these peaks, a tuple $(z_{p,k}, c_{p,k})$ is stored, where $z_{p,k}$ is the depth and $c_{p,k}$ the NCC score for the k^{th} peak of pixel p. The MRF formulation of (2.9) is then optimized using these peaks. $K+1$ labels are used in the optimization: one for each of the possible K peaks, and an additional label, U, identitfying the unknown state. They then define the data term as:

$$c(l_p) = \begin{cases} \lambda e^{-\beta c_{p,k}} & l_p \in [1 \dots K] \\ \phi_U & l_p = U \end{cases}, \tag{2.16}$$

where ϕ_U represents a fixed cost for selecting the unknown label, and λ, β are tunable parameters. The smoothness term is defined as:

$$V_{pq}(l_p, l_q) = \begin{cases} 2\frac{|z_{p,l_p} - z_{q,l_q}|}{z_{p,l_p} + z_{q,l_q}} & l_p \in [1 \dots K] & l_q \in [1 \dots K] \\ \psi_U & l_p \in [1 \dots K] & l_q = U \\ \psi_U & l_p = U & l_q \in [1 \dots K] \\ 0 & l_p = U & l_q = U \end{cases}, \tag{2.17}$$

where ψ_U is a tunable pairwise cost when just one of the labels is U. The authors suggest setting ψ_U to a small value so as to encourage larger regions of pixels to be labelled as unknown. This can reduce a large amount of noise that is normally

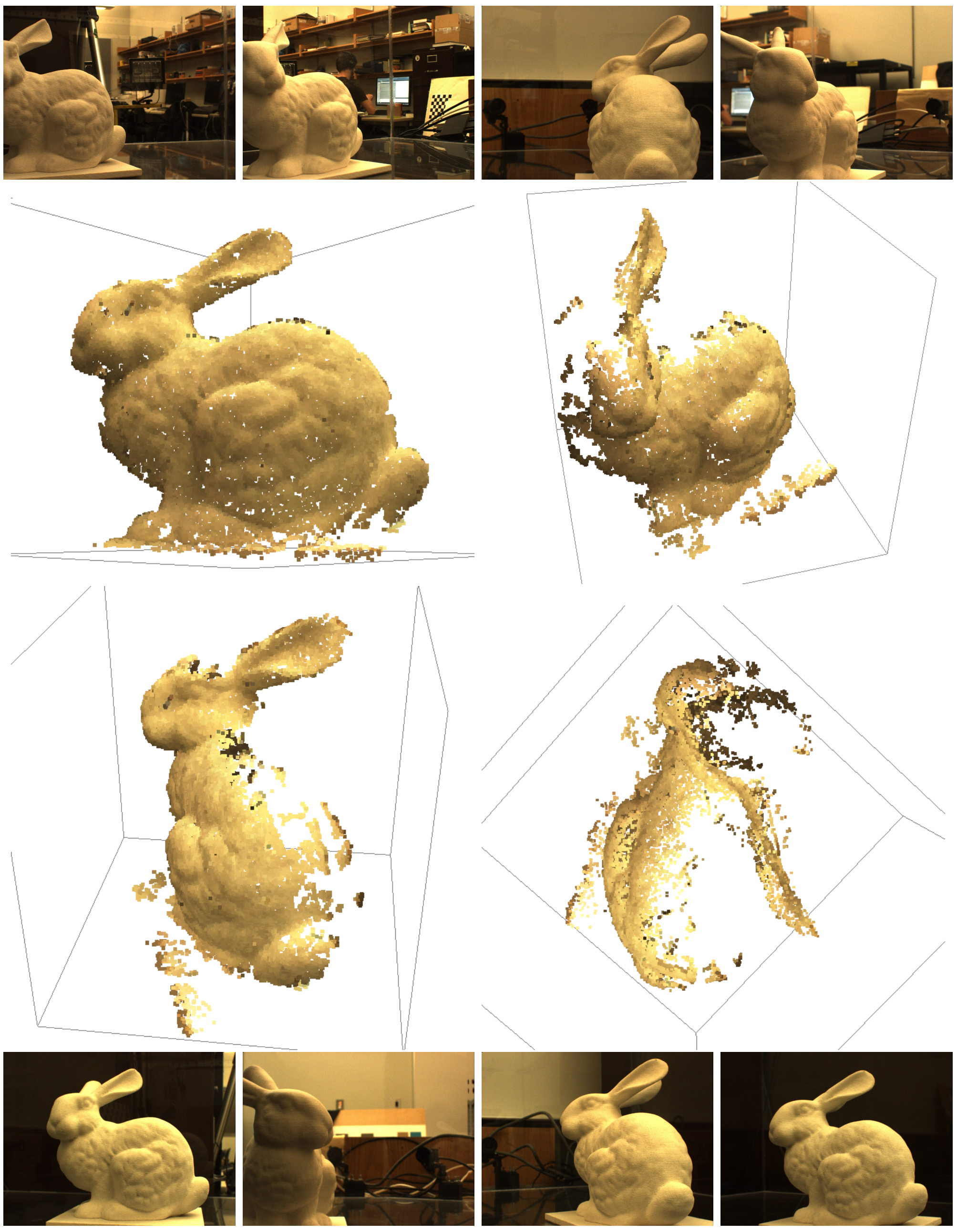


Figure 2.12: Using eight views of a Stanford bunny model (top and bottom rows), four different views of the reconstructed patches using the implementation provided by Furukawa and Ponce [13].

seen in regions of matching failure. Figure 2.13 shows various results achieved from our implementation of the method by Campbell et al., with parameters $K = 9, N = 3, \beta = \lambda = 1, \phi_{\mathrm{U}} = 0.5, \psi_{\mathrm{U}} = 0.002$, and a 5×5 window for NCC. In particular, it shows how a larger number of depth hypotheses can be obtained by intelligently incorporating additional views.

2.3 Underwater Stereo

Underwater stereo vision is an area that has received a relatively low amount of attention, but thhere are several significant works in the area. We separate this section into related works of the two most relevant components of our system: calibration and stereo matching.

2.3.1 Calibration

Calibration of an underwater stereo system is extremely important. As shown by Treibitz et al., assuming a single viewpoint (SVP) model, i.e., improperly modelling refraction, can be very erroneous [46]. The most prominent issue with existing methods of underwater calibration (and underwater stereo matching), is the assumption that cameras share the same refractive interface, a potentially prohibitive assumption. This model is applicable in many cases, but there are also many cases where it is not. For example, in Figure 1.1 each camera has its own housing, which in turn means that each camera has a different refractive interface.

An early work in calibration of an underwater stereo system is given by Li et al. [30]. They place a stereo pair inside a waterproof housing where the cover lens is hemispherical inside the housing and planar on the outside. They derive a set of linear equations that can be optimized using least squares when a known set of 3D points (in object space) are given. The process is separated into two phases. In the first phase, traditional intrinsic (focal length, lens distortion, etc.) and extrinsic (rotational + translation) parameters of the system are calibrated. In the second phase, the cover lens parameters and refractive indices are obtained. The argument for separating these two calibrations is that refractive distortion may accidentally be "absorbed" into the lens distortion parameters, resulting in an incorrect calibration.

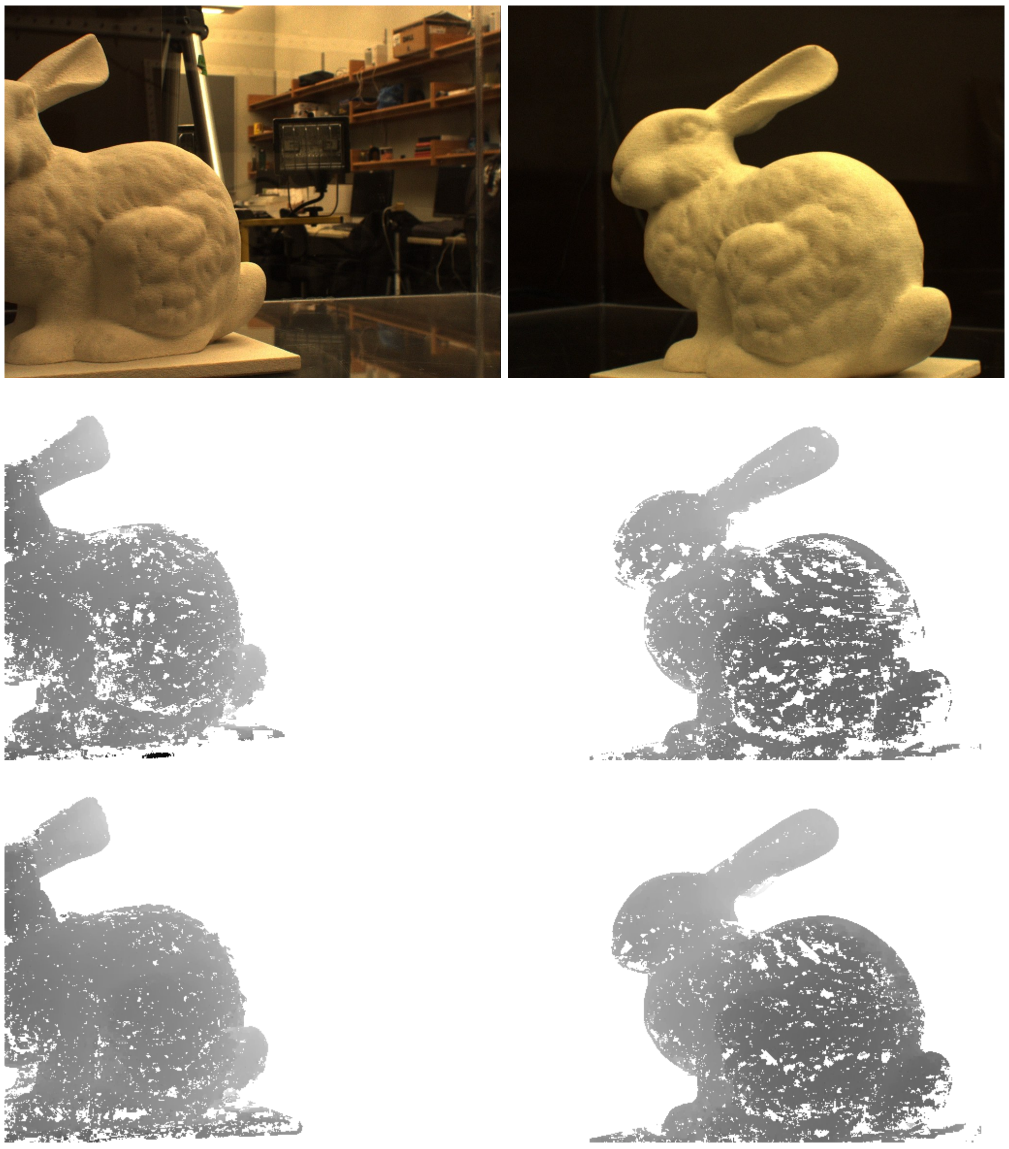


Figure 2.13: Example results from our implementation of the method of Campbell et al. [7]. Two different views of the Stanford bunny model (top row) and the resulting depth maps produced when using three (middle row) and eight views (bottom row). More views provide better peaks, which reduces the number of pixels being labelled as "unknown" (white pixels).

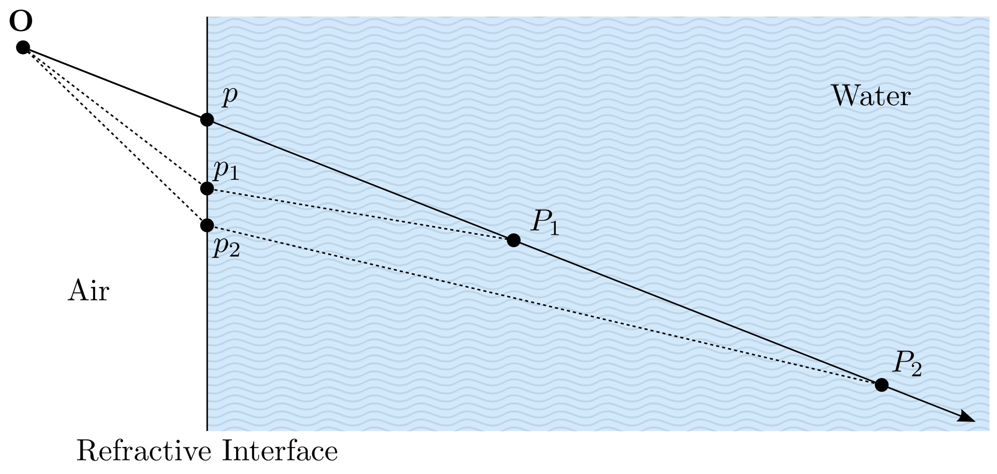


Figure 2.14: Refractive distortion is dependent on the 3D location of a point. Two different points on a ray, P_1 and P_2, are projected onto an interface. Assuming no refraction, both of these points project to p. Given a refractive interface, P_1, P_2 project to points p_1, p_2, respectively. These two points have different radial offsets, hence it is erroneous to model this as radial lens distortion.

Various other calibration methods completely ignore refraction, assuming that its impact is purely radial, hence allowing it to be calibrated alongside radial lens distortion [28, 33, 40]. Although improvements can be acquired [28], these approaches fail to take into consideration that the *amount* of radial distortion varies based on the 3D location of points in the scene (Figure 2.14). Unfortunately, lens distortion only considers the 2D imaging location of points, so it cannot properly compensate for refractive distortions. Yet, there are scenarios where a lens distortion model could compensate for refractive distortion. Consider a scene where the range of depths is very narrow. Intuitively, the dependence on depth is minimal in such a scenario, so lens distortion models could compensate for most of the refractive distortion. This is a very restrictive scenario, and we prefer to tackle the problem in general.

Modeling a single planar refractive interface requires three parameters: two for the normal, and one for distance (Figure 2.15). A simple approach one might take to calibrate the parameters of the interface would be to place a calibration pattern on the interface itself, similar to the floating pattern used in [12]. If the interface is relatively distant from the camera, this is indeed a straightforward and simple approach to find interface parameters directly from existing calibration techniques. Unfortunately, this is generally impractical for cameras in small, watertight housings.

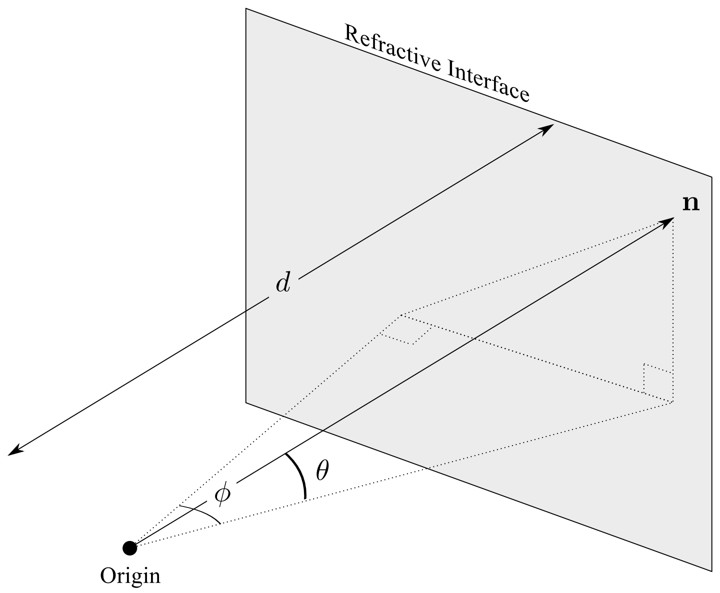


Figure 2.15: Modelling a planar refractive interface. Three parameters are necessary to define a plane: angles θ and ϕ define the normal, $\mathbf{n}$, and d specifies the distance from the origin to the plane, in the direction of the normal.

A recent study in the importance of modelling refraction was presented by Treibitz et al. [46]. They argue that assuming an SVP model, even with some form of correction, can be very erroneous in systems with a flat refractive interface. They present a means of calibrating a *ray map*, that is, a set of pixel/ray correspondences such that any point on a given ray will project to its corresponding pixel. The downside of their calibration method is that the planar refractive interface has to be perpendicular to the optical axis.

Finally, Kunz and Singh investigate the error of planar and hemispherical interfaces under simulated conditions [26]. The advantage of a hemispherical interface is that, if the center-of-projection of a pinhole camera is placed directly at the center of the hemisphere the effects of refraction are completely eliminated. Such placement can be difficult and may not always be practical. They also show that radial lens distortion models can perhaps compensate for a large portion of refractive distortion, but only for cases in which the camera's optical axis forms small angles with the normal of the refractive interface.

Although not directly related to calibration, the work of Morris and Kutulakos has some significance [31]. They present a method of finding the positions and normals of an arbitrary refractive surface. Knowing this information allows us to compute the refracted ray for each pixel, but projection into a second view becomes more difficult. To alleviate this problem, the authors choose to use an easily detectable pattern to establish correspondence between two views. Hence, this approach is not quite suitable for general underwater stereo. We believe that choosing a simple geometric model for the refractive interface provides both a simpler and more elegant solution to this issue, as will be seen in Chapter 3.

2.3.2 Stereo Matching

Underwater stereo matching is an area that has received little attention. Perhaps one reason for this is that radial lens distortion models are believed to sufficiently compensate for refractive distortion and hence, traditional stereo methods can be readily applied. Also, depending on the stereo setup, it is possible that traditional epipolar geometry is sufficient [54].

Queiroz-Neto et al. take a photometric approach to underwater stereo [36]. In particular, they acknowledge the impact of light attenuation and scattering in underwater imaging. Their focus is the removal of these effects to improve hypothesis costs in regular stereo, so refraction is completely ignored. Although they show improvements for matching, it is likely that their experiments presented little refractive distortion. Also, since they used a global algorithm, the lack of detail in their scenes perhaps allow the smoothing terms to give nice results. In other words, a more complex scene and setup will require additional corrections beyond radiometric correction.

Other methods consider high-level approximations to a physically-inspired model. Snell's Law relates a ray's incident angle, θ_i, with its refracted angle, θ_r, as such:

$$n_i \sin\theta_i = n_r \sin\theta_r, \tag{2.18}$$

where n_i, n_r are known as the *refractive indices* for the incident and refractive mediums, respectively. For example, water has a refractive index of approximately 1.333. Ferreira et al. [12] use Snell's Law to relate an incident ray $v_i = \begin{bmatrix} v_i^x & v_i^y & v_i^z \end{bmatrix}^{\mathrm{T}}$, with its refracted ray, v_r:

$$v_r = \begin{bmatrix} n \cdot v_i^x \\ n \cdot v_i^y \\ -\sqrt{(1-n^2)\left((v_i^x)^2 + (v_i^y)^2\right) + (v_i^z)^2} \end{bmatrix}, \tag{2.19}$$

where $n = \dfrac{n_i}{n_r}$. A first-order Taylor series approximation of (2.19) (in an appropriate neighbourhood) shows that:

$$v_r \approx \begin{bmatrix} n \cdot v_i^x \\ n \cdot v_i^y \\ v_i^z \end{bmatrix}. \tag{2.20}$$

Equation (2.20) manifests itself as a scaling of a camera's focal length, which has been previously observed, for example, by Lavest et al. [28]. As Ferreira et al. point out, this does not truly fix the problem, but rather hides it somewhat. To compensate for refractive distortion, a two-dimensional search centered on the epipolar line is performed. The larger number of window comparisons made increases the computationally complexity and also increases the potential for establishing false positive matches. Nevertheless, once a match is established they triangulate a 3D point without any approximation. This latter point is important because, as we will show in Chapter 5, ignoring refraction for stereo matching *and* triangulation has a twofold impact on error.

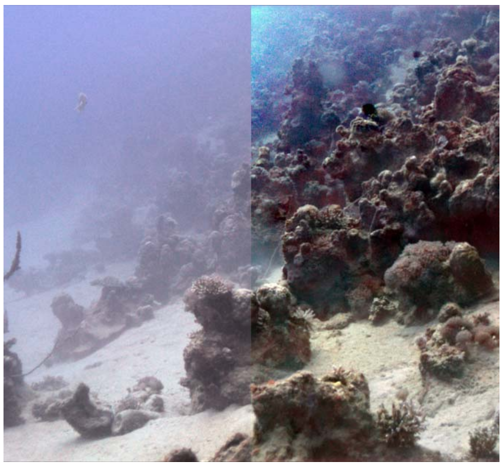
Perhaps one of the more significant works in the area of underwater stereo vision is the theoretical analysis of refractive epipolar geometry given by Chari and Sturm [9]. Given the nonlinear nature of refraction, establishing this geometry requires a notion of *lifted coordinates*. As the name suggests, lifted coordinates involves "lifting" a vector to a higher-dimensional space. In particular, given a vector $\mathbf{v} = \begin{bmatrix} \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \end{bmatrix}^{\mathrm{T}}$ the lifted vector $\hat{\mathbf{v}}$ is:

$$\begin{aligned} \hat{\mathbf{v}} &= \begin{bmatrix} \mathbf{v}_x^2 & \mathbf{v}_x\mathbf{v}_y & \mathbf{v}_y^2 & \mathbf{v}_x\mathbf{v}_z & \mathbf{v}_y\mathbf{v}_z & \mathbf{v}_z^2 \end{bmatrix}^{\mathrm{T}} \\ &= \begin{bmatrix} \hat{\mathbf{v}}_1 & \hat{\mathbf{v}}_2 & \hat{\mathbf{v}}_3 & \hat{\mathbf{v}}_4 & \hat{\mathbf{v}}_5 & \hat{\mathbf{v}}_6 \end{bmatrix}^{\mathrm{T}} \end{aligned}. \tag{2.21}$$

Through a series of derivations, Chari and Sturm are able to derive an equation that resembles that of the traditional epipolar constraint. Given a point correspondence $\{\mathbf{u}, \mathbf{v}\}$, the following condition is satisfied:

$$\begin{bmatrix} \hat{\mathbf{v}} \\ \hat{\mathbf{v}} \cdot \mathbf{v}_z^2 \end{bmatrix}^{\mathrm{T}} \mathcal{F} \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{u}} \cdot \mathbf{u}_z^2 \end{bmatrix} = 0, \tag{2.22}$$

where $\mathcal{F}$ is a 12×12 matrix, the *refractive fundamental matrix* [9]. The equation $\mathcal{F} \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{u}} \cdot \mathbf{u}_z^2 \end{bmatrix} = 0$ represents a quartic curve, the refractive epipolar curve for $\mathbf{u}$. In any case, Chari and Sturm's derivations are for a shared interface. As we have argued before, this model is not always applicable. It would be interesting to see if this framework could be extended to multiple interfaces, although such a direction is not taken in this thesis.


(a) Corrected/uncorrected image

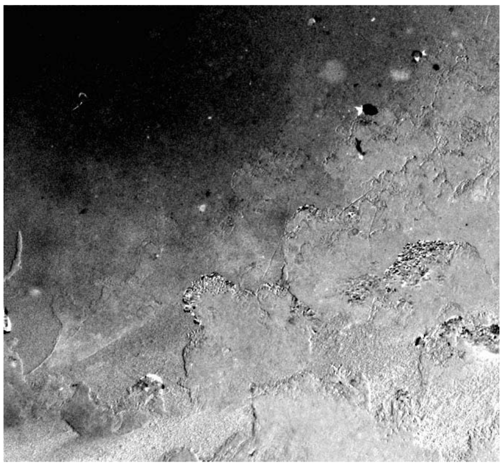

(b) Range image

Figure 2.16: (a) Schechner and Karpel [38] use a polarizing lens and a detailed image formation model to reduce the impact of attenuation and scattering. (b) With their image formation model depths can be approximated. (Source: [38])

2.3.3 Other Methods

Although not a stereo matching method, it is of interest to introduce the work of Schechner and Karpel [38]. They dig into the image formation process, modelling attenuation and backscattering effects. By installing a polarizer on their cameras, they can dramatically improve the quality and viewing range of underwater images (Figure 2.16(a)). Although not the primary intent, they are also able to estimate depths from their correction (Figure 2.16(b)). Unfortunately, they suggest a hemispherical interface, instead of a planar one, and that the center of projection of the camera is at the center of this hemisphere. As mentioned previously, this is less practical and difficult to achieve.

The previous section dealt with stereo methods which use regular images and epipolar geometry to establish correspondence. This is commonly referred to as *passive* stereo. Another approach to the correspondence problem is to project controlled patterns of light onto the scene. These methods are called *structured light methods* and fall under what is known as *active* stereo.

One issue in typical passive stereo techniques is that of matching ambiguity. This occurs, for example, in occluded and textureless areas (no clear match) and for repeated texture (multiple good matches). Structured light methods overcome these ambiguities by introducing additional information into the scene (i.e., projected light patterns).

Unfortunately, many structured light methods have the downside that multiple patterns have to be projected, or several images must be grabbed over a period of time. This means that many structured light methods are not appropriate for highly dynamic scenes, but there are various structured light methods that can perform a "one-shot" approach (e.g., de Bruijn patterns [47], colored patterns [56]).

Various structured light methods have been proposed for underwater stereo. Narasimhan and Nayar [32] focus on the analysis of scattering media and its impact on structured light methods. They introduce two methods: light stripe scanning and photometric stereo. Light stripe scanning involves sweeping a plane of light across a scene. The intersection of this plane of light and surfaces of objects results in a curve, which can be detected and used to simplify the correspondence problem. Light stripe scanning is a slow process, which makes it only useful in static scenes.

Photometric stereo takes a different angle on reconstruction: finding surface normals[4]. Three images are sufficient for reconstructing surface normals in the absence of scattering, and five are sufficient when scattering is present [32]. Narasimhan and Naya formulate their work based on an orthographic camera, in which case – as long as the refractive interface is parallel to the imaging plane – refraction plays very little role. The perspective pinhole model is more common when modelling a typical camera.

Kawai et al. use a *space encoding* method to establish correspondence. In particular, space-encoding methods project various patterns onto a scene (Figure 2.17). These patterns light up points of a scene differently, in such a way that each point obtains a unique signature that can be used for simplifying the search for correspondence. Given this, highly dynamic scenes are still problematic since multiple patterns need to be projected.

Swirski et al. [42] observe a natural phenomenon that can provide the same disambiguation power as space-encoding methods: caustics. Caustics are naturally occurring, random patterns that illuminate the scene. Underwater caustics, such as those in a pool, are often seen as "web-like" patterns of bright light (Figure 2.18). These areas of bright light are simply areas where light becomes

[4] By integrating over a set of normals, one can reconstruct a surface.

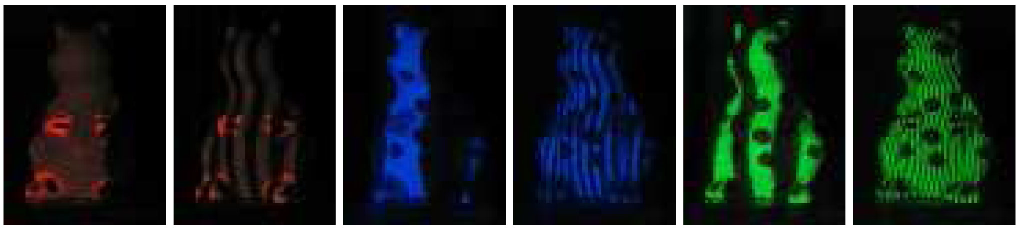


Figure 2.17: Binary patterns used by the method of Kawai et al. [22]. These patterns allow one to encode a pixel with a unique identifier. Correspondence then becomes the task of finding a geometrically consistent pixel with the same identifier. (Source: [22])

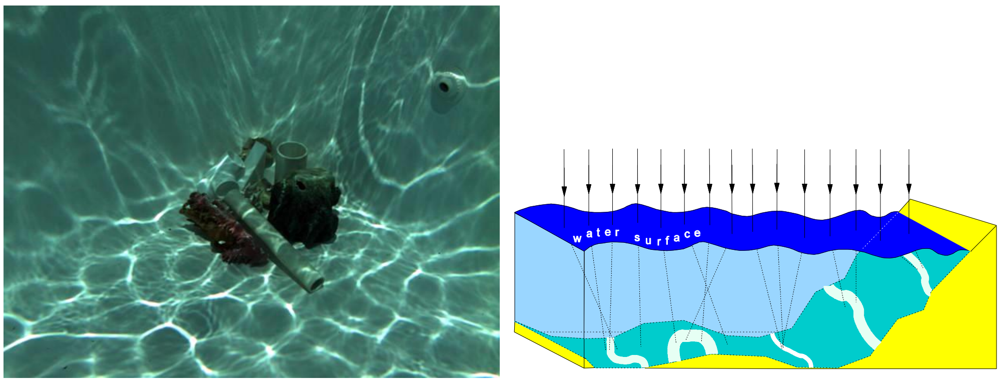


Figure 2.18: Caustics illuminate the bottom of a pool (left). Caustics are formed when light rays are concentrated in an area (right). This concentration of light rays is often caused by a refractive/reflective surface. (Source: [42])

concentrated due to an irregular refractive surface. Given a refractive surface that is disturbed, the variation in the surface's shape will change the caustics over time. These temporally varying regions of brightness give us the ability to disambiguate stereo matches, similar to structured light methods.

Swirski et al. show how, given enough frames, they can establish correspondence with a high degree of confidence. In particular, they aggregate the pixel intensities over time into a single vector, and perform normalized cross correlation. Let $\overline{\mathbf{X}}$ be the mean vector of $\mathbf{X}$, that is, a vector in which every element is equal to the average of all elements in $\mathbf{X}$. Let

$$\widehat{\mathbf{X}} = \mathbf{X} - \overline{\mathbf{X}} \tag{2.23}$$

be the mean-subtracted vector of $\mathbf{X}$. The normalized cross-correlation [29] of vectors $\mathbf{A}$ and $\mathbf{B}$ is defined as

$$NCC(\mathbf{A}, \mathbf{B}) = \left\langle \frac{\widehat{\mathbf{A}}}{\| \widehat{\mathbf{A}} \|} \frac{\widehat{\mathbf{B}}}{\| \widehat{\mathbf{B}} \|} \right\rangle, \tag{2.24}$$

where $\langle \cdot \rangle$ is the inner product of two vectors and $\| \cdot \|$ the L_2 norm [53].

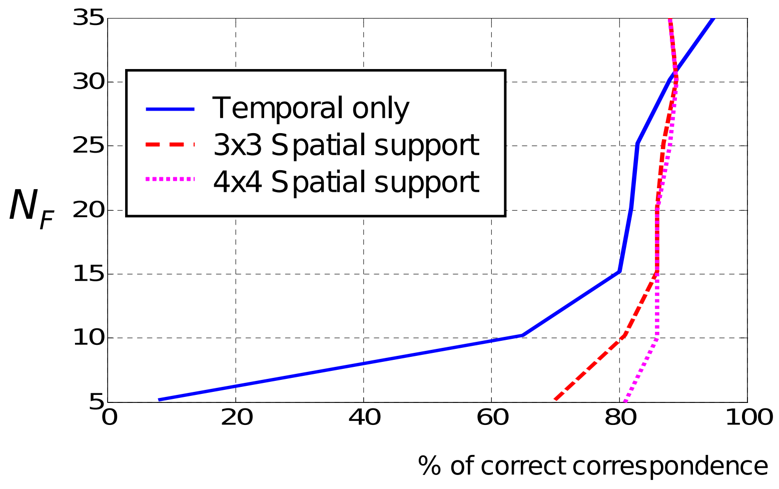


Figure 2.19: The number of frames required versus the rate of true positive matches when using the method of Swirski et al. [42]. By incorporating a small support window around a pixel (i.e., spatiotemporal correlation), a much smaller number of frames can be used to achieve a high correspondence rate when compared to just temporal correlation. (Source: [42])

Now, let $I_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, j)$ be the intensity of pixel $\mathbf{x}_{\mathrm{L}}$ in the j^{th} frame of the left image. Denote

$$\mathbf{I}_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}) = \begin{bmatrix} I_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, 1) \\ I_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, 2) \\ \vdots \\ I_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, N) \end{bmatrix}, \tag{2.25}$$

the aggregation of the intensities of pixel x_{L} over N frames. Consider a candidate match, $\mathbf{x}_{\mathrm{R}}$, in the right image with $\mathbf{I}_{\mathrm{R}}(\mathbf{x}_{\mathrm{R}})$ defined similarly as in (2.25). Swirski et al. call $NCC\left(\mathbf{I}_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}), \mathbf{I}_{\mathrm{R}}(\mathbf{x}_{\mathrm{R}})\right)$ as the *temporal* correlation, as it correlates the same pixel over time. To reduce the number of frames required, they suggest using spatio-temporal correlation as an alternative. That is, correlate not only the same pixel over time, but include the support of neighbouring pixels as well (Figure 2.19).

This method shares the same issues as structured light methods that project multiple patterns onto the scene. In particular, there is the issue of dynamic scenes; if caustics are available and the scene is static, the approach taken by Swirski et al. is quite effective. Yet, there are many situations when caustics may not be available, or are of insufficient intensity to provide reliable results. These situations include:

- any parts of the scene that lie in the shadow of an object from above (as suggested by Swirski et al.),
- nighttime scenes, due to the lack of lighting, and
- at depths where light is mostly attenuated.

In these situations another approach would have to be taken. In any case, our refractive epipolar geometry can still provide extra support for this approach, by restricting the domain of candidate pixels to search through.

Chapter 3

Refractive Imaging

In this chapter we begin by introducing the concepts of the *pinhole camera model* and *epipolar geometry*. We follow this with the mathematical foundation necessary to understand the behaviour of refraction. We reformulate image projection under the influence of a planar refractive interface so that it can be used in a stereo matching framework. Using these formulations we show how traditional epipolar geometry is no longer valid, and how captured images no longer have a single center-of-projection.

3.1 Image Formation

3.1.1 Pinhole Camera Model

Modern camera lenses are incredibly complex, often being the composite of several intermediate lenses (Figure 3.1). The interactions of these lenses helps to reduce various undesirable effects, such as lens distortion and chromatic aberration.

Modelling such a lens mathematically is difficult for computer vision research[1], so many people model the system as a *pinhole camera*. A pinhole camera considers all light rays that pass through a single point in space (Figure 3.2), known as the *center-of-projection* (CoP). Suppose the pinhole camera has a focal length of f, that is, the camera's imaging plane is at a distance f from the CoP. Given a point $\mathbf{x} = \begin{bmatrix} \mathbf{x}_x & \mathbf{x}_y & \mathbf{x}_z \end{bmatrix}^\mathrm{T}$, we can find its projection to pixel p using similar triangles:

$$\frac{f}{\mathbf{x}_z} = \frac{p_x}{\mathbf{x}_x}, \tag{3.1}$$

[1] More complex models exist, such as thick/thin lens models.

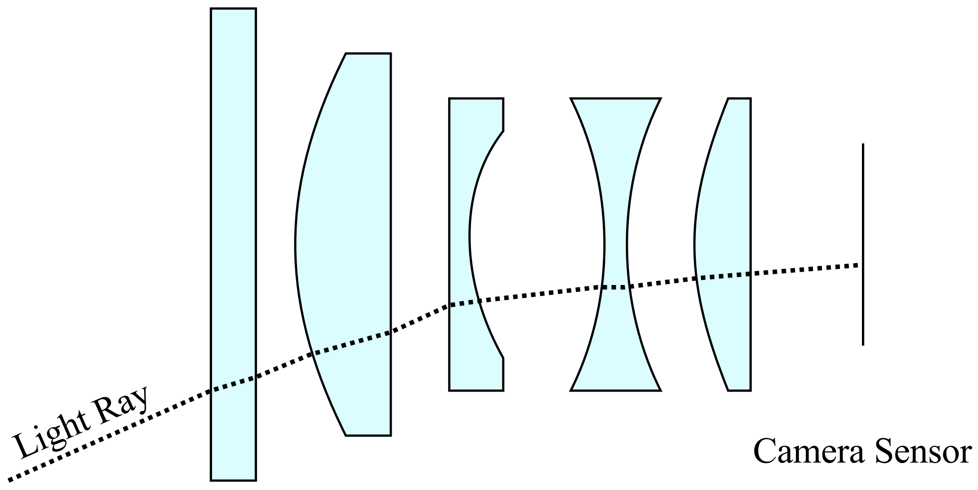


Figure 3.1: The nature of a modern camera lens. Lenses are often composed of multiple sub-lenses that act together to produce a higher quality image.

and similarly for p_y. We can formulate this as a matrix product:

$$\hat{p} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}, \tag{3.2}$$

where $\hat{p} = \begin{bmatrix} \hat{p_x} & \hat{p_y} & \hat{p_z} \end{bmatrix}^{\mathrm{T}}$ is the homogeneous representation of pixel p. The image coordinates of p can be obtained from $\hat{p}$:

$$p = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \frac{1}{\hat{p_z}} \begin{bmatrix} \hat{p_x} \\ \hat{p_y} \end{bmatrix}. \tag{3.3}$$

3.1.2 Pixel Coordinates

Section 3.1.1 outlines the basis for projection in the pinhole camera. It is common to map projected points into image-space, i.e., pixel-based coordinates. For example, (3.2) does not take into consideration the fact that pixels in an image are often contained in $[0, w] \times [0, h]$, where w, h are the width and height of the camera, respectively. There are typically five parameters that take a 3D point and map it into image-space [19]:

- σ_x, σ_y, scale factors in the x, y-coordinate directions, respectively[2],
- s, the pixel skew, and

[2] Note that σ_x and σ_y can be regarded as the focal length multiplied by scale factors that convert regular units into image-space units.

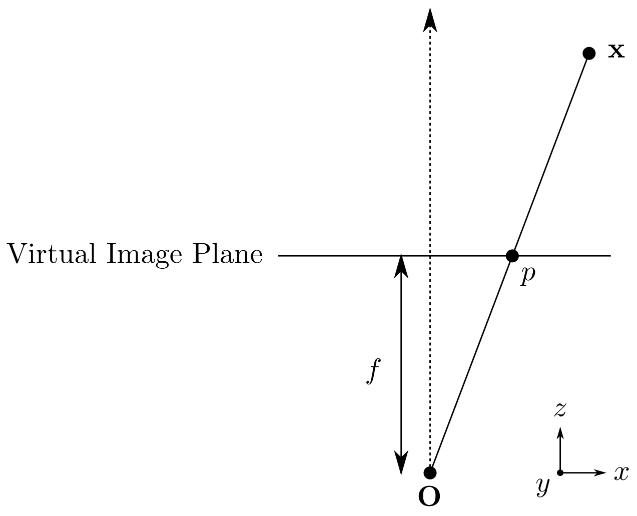


Figure 3.2: The pinhole camera model. A 3D point $\mathbf{x}$ is projected to point p, where the camera has a focal length of f.

- $\begin{bmatrix} x_0 & y_0 \end{bmatrix}^\mathrm{T}$, pixel coordinates of the principal point.

These five parameters are the *intrinsic parameters* of the camera, and can be combined into a 3×3 matrix:

$$K = \begin{bmatrix} \sigma_x & s & x_0 \\ 0 & \sigma_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.4}$$

known as the *intrinsic matrix*. Projection into image-space then is simply matrix multiplication:

$$\hat{p} = K\mathbf{x}. \tag{3.5}$$

Note that (3.1–3.5) deal with points in a frame of reference such that the camera's CoP is at the origin and x, y, z-axes correspond to image x, y axes and the viewing direction, respectively. We can generalize this to different coordinate bases by incorporating a rotation and a translation that bring a 3D point into the camera's local basis. For the rest of this thesis we will refer to the camera's frame of reference as the *local space* and a general frame of reference as the *global space*. Given a rotation matrix R and a translation vector t, we can translate $\mathbf{x}$ from global space to local space:

$$\mathbf{x}' = R\mathbf{x} + t = \begin{bmatrix} R & t \end{bmatrix} \hat{\mathbf{x}}, \tag{3.6}$$

where $\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x}^\mathrm{T} & 1 \end{bmatrix}^\mathrm{T}$ is the homogeneous representation of $\mathbf{x}$, and $\mathbf{x}'$ is the local space representation of $\mathbf{x}$.

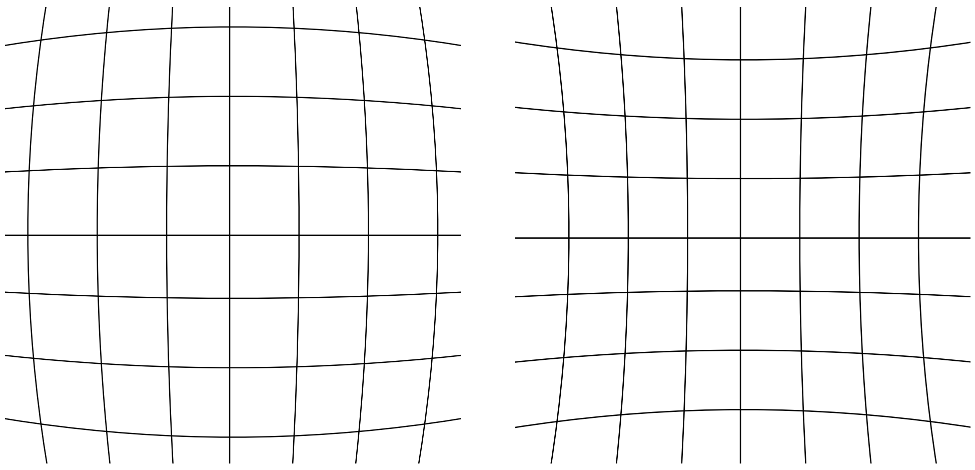

(a) Barrel Distortion (b) Pincushion Distortion

Figure 3.3: Two common forms of lens distortion. Barrel distortion "pushes" pixels away from a central point. Pincushion distortion "draws" pixels towards a central point.

Putting (3.5) and (3.6) together we obtain the pinhole projection model:

$$\hat{p} = K \begin{bmatrix} R & t \end{bmatrix} \mathbf{x} = Px, \tag{3.7}$$

where P is the *projection matrix*.

3.1.3 Lens Distortion

Section 3.1.2 provides a means of bringing projected points into image-space coordinates, but only under an ideal lens. Unfortunately, given the complex nature of lenses (Figure 3.1), distortions are often present. This is due to the refraction that occurs when light strikes the lens, and is more significant in lenses with smaller focal lengths [19] since higher degrees of bending will occur. Figure 3.3 shows two of the simpler forms of distortion: barrel and pincushion.

To project points to the correct pixel it is important to compensate for these distortions. In this thesis, OpenCV [2] was used for modelling distortion. Given pixel $p = \begin{bmatrix} p_x & p_y \end{bmatrix}^{\mathrm{T}}$ from the local space projection in (3.3), OpenCV uses a high-order polynomial model:

$$\begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = \begin{bmatrix} p_x \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2t_1 p_x p_y + t_2 \left(r^2 + 2p_x^2\right) \\ p_y \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2t_2 p_x p_y + t_1 \left(r^2 + 2p_y^2\right) \end{bmatrix}, \tag{3.8}$$

where $r = \sqrt{p_x^2 + p_y^2}$, k_i are radial distortion parameters, t_i are tangential distortion parameters, and $p' = \begin{bmatrix} p'_x & p'_y \end{bmatrix}^{\mathrm{T}}$ is the undistorted pixel. This is just one of

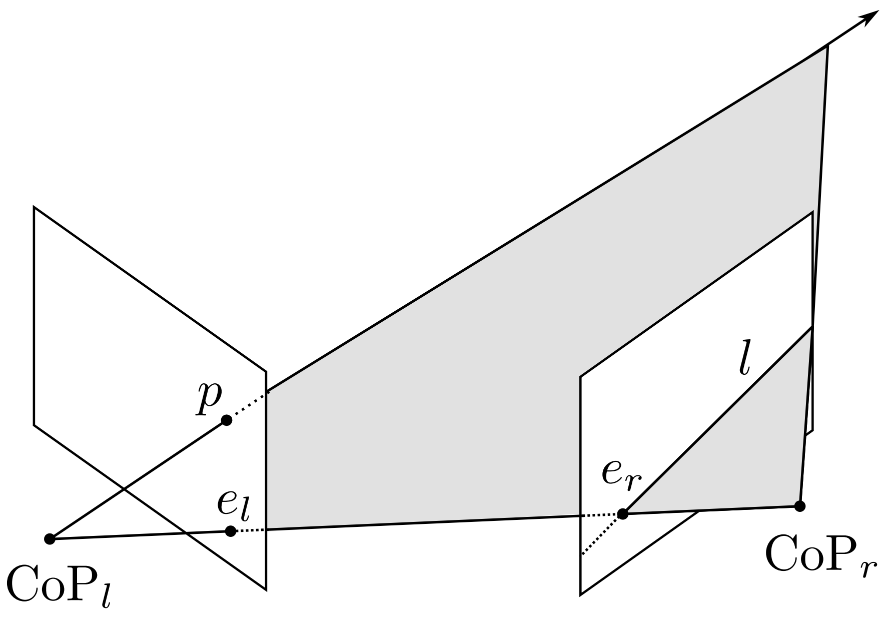


Figure 3.4: The epipolar geometry of two cameras. The pixel p in one image is restricted to the epipolar line l in another. All epipolar lines most go through their corresponding epipoles (e_l and e_r). The epipoles are the intersections of the line connecting the two centers-of-projection (CoP_l and CoP_r) with the corresponding image planes.

many proposed models of lens distortion (e.g., [10]). Once the radial/tangential parameters are estimated, it is common to process an image to eliminate lens distortion [19].

3.2 Epipolar Geometry

The correspondence problem is a fundamental problem in stereo vision, but it has proven to be a challenge. In the most general case, a feature in one image requires a two-dimensional search in another image to establish correspondence. This makes the problem computationally complex and challenging, but a stereo pair forms an intrinsic geometry that allows us to reduce this problem to a one-dimensional search. This geometry is known as the *epipolar geometry* of the pair (Figure 3.4).

Of particular importance is the *epipolar constraint*. Given an image point correspondence $\{\hat{p}, \hat{p}'\}$ the following holds:

$$\left(\hat{p}'\right)^{\mathrm{T}} F \hat{p} = 0, \tag{3.9}$$

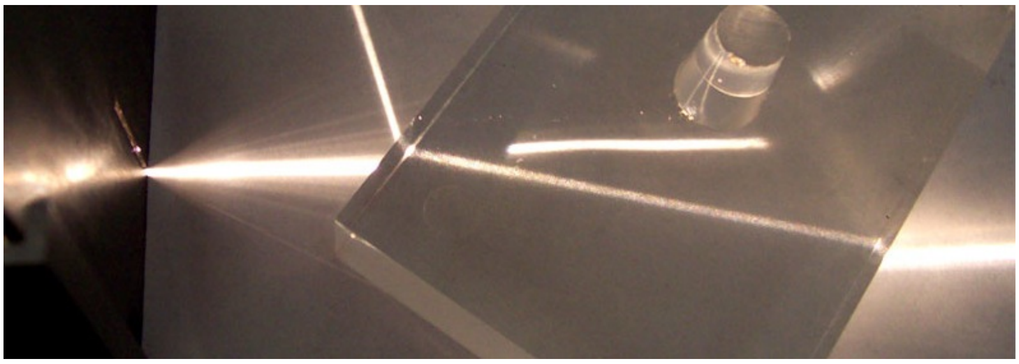


Figure 3.5: Illustrating refraction in a block of Perspex. (Source: http://en.wikipedia.org/wiki/File:Refraction.jpg)

where F is a 3×3 matrix which relates pixels in the two views, known as the *fundamental matrix*. In particular, given a fixed $\hat{p}$, the set of all points $\hat{p}'$ that satisfy (3.9) form a line in one image, known as the *epipolar line*. This is the essence of epipolar geometry. Given a point in one image, $\hat{p}$, we can reduce the search for the corresponding point on the corresponding epipolar line.

3.3 Refraction

Refraction (of light) is the change in direction of light when it changes speed (Figure 3.5). This change in speed can occur for many reasons, but is mostly associated with instances where light moves from one medium to another with a different *refractive index*. A medium's refractive index is a measure of the speed of light in that medium versus the speed of light in a vacuum. For example, the refractive index of water is approximately 1.333, which means light travels 1.333 times faster in a vacuum than it does in water

Snell's Law is a fundamental relation between the angles of incidence and refraction of light. Given angles of incidence and refraction, θ_i and θ_r respectively, and refractive indices n_i/n_r of the incident/refractive medium (Figure 3.6), Snell's Law states:

$$n_i \sin \theta_i = n_r \sin \theta_r. \tag{3.10}$$

It is important to be able to compute the refracted ray from a given ray and a refractive interface. Suppose $\vec{\mathbf{n}}$ is the normal of the surface and $\vec{d_i}$ the direction of the incoming ray, then the direction of the refracted ray, $\vec{d_r}$, can be found

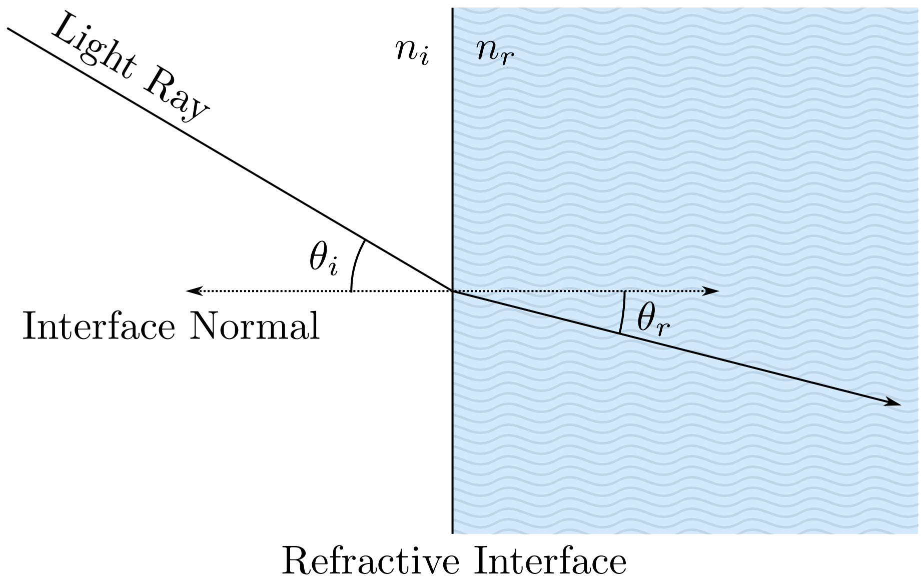


Figure 3.6: The theory of refraction is given by Snell's Law (3.10).

by [16]:

$$\cos\theta_i = \vec{\mathbf{n}} \cdot (-\vec{d_i}) \tag{3.11}$$

$$\cos\theta_r = \sqrt{1 - \left(\frac{n_i}{n_r}\right)^2 (1 - \cos^2\theta_i)} \tag{3.12}$$

$$\vec{d_r} = \left(\frac{n_i}{n_r}\right)\vec{d_i} + \left(\frac{n_i}{n_r}\cos\theta_i + cos\theta_r\right)\vec{\mathbf{n}} \tag{3.13}$$

For most underwater imaging applications, cameras are placed inside waterproof housings filled with air, in which case $n_i \approx 1$. Hence, for the rest of this thesis we will write n_r as n for simplicity.

3.4 Refractive Projection

Since a refractive interface will bend light in a nonlinear fashion, the traditional projection formula given in (3.7) is no longer valid. Hence, results will be erroneous unless we physically model this bending [46]. We also neglect a glass window between air and water (Figure 3.7) because a thin glass panel does not change a ray's final direction, but rather induces a radial shift (Figure 3.8) that is generally negligible with regards to the distances of the scene objects [46].

For example, the tank in our lab has a glass thickness of approximately 5 mm. Given a camera with a field-of-view of 70° both horizontally and vertically, the most extreme radial shift of a ray (in the image corners) is 0.45 mm, a value

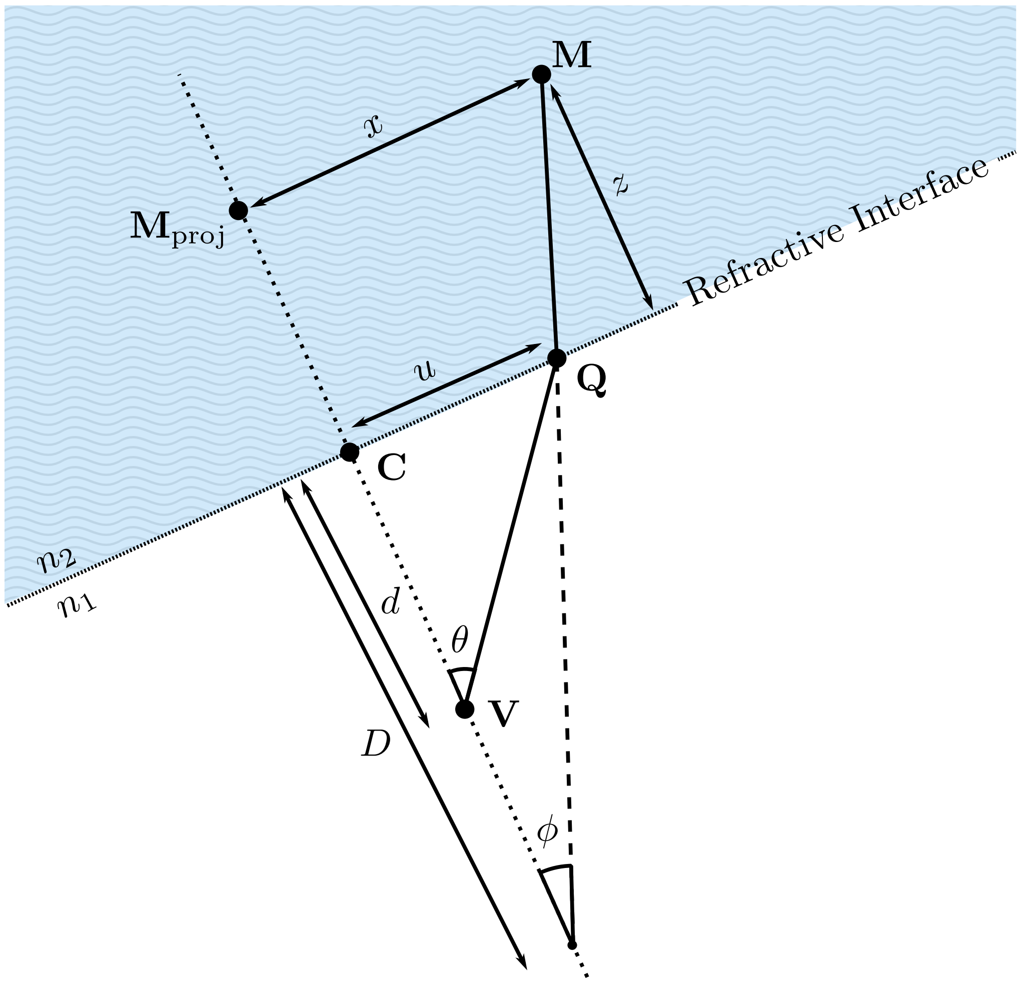


Figure 3.7: Cross section of a camera system centred at **V** and under the influence of a planar refractive interface. When projecting points into the camera, the only unknown is u.

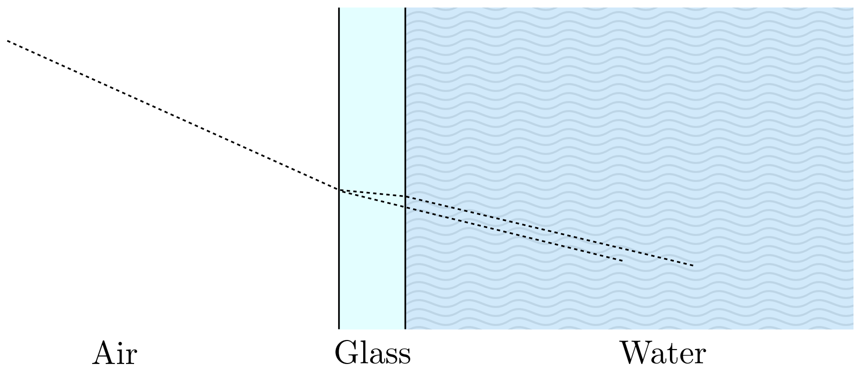


Figure 3.8: When a glass window is introduced between the air-water interface, the final direction of the ray is the same as without the panel, but is shifted radially.

that was computed through simulation. Approximately 180 mm of an imaged object covered 768 pixels, which means that there are roughly 4.3 pixels/mm. This equates to a pixel offset of just under a pixel when working with half-resolution images. This is the worst case offset, occurring in the corners of an image. From our own experiments (Chapter 5), the impact has not been significant enough to consider, and our method still provides superior results to others that approximate refractive distortion. However, given a glass window of sufficient thickness or a wide-angle lens, modelling such a radial shift would be necessary.

Given the setup in Figure 3.7 we have:

$$u = D \tan\phi \tag{3.14}$$

Using Snell's Law, we can derive the following through trigonometric identities:

$$\tan^2\phi = \frac{\sin^2\phi}{1-\sin^2\phi} = \frac{\sin^2\theta}{n^2-\sin^2\theta}, \tag{3.15}$$

$$\sin^2\theta = \frac{\tan^2\theta}{1+\tan^2\theta} = \frac{u^2}{u^2+d^2}. \tag{3.16}$$

Substituting (3.16) into (3.15) and simplifying we obtain:

$$\tan^2\phi = \frac{u^2}{n^2d^2+(n^2-1)u^2}. \tag{3.17}$$

Substituting (3.17) into (3.14) we arrive at:

$$D = \sqrt{n^2d^2+(n^2-1)u^2}. \tag{3.18}$$

Now suppose we want to find the projection, $\mathbf{Q}$, of $\mathbf{M}$ onto the refractive interface given a viewpoint $\mathbf{V}$. Using similar triangle ratios we obtain:

$$\frac{D}{u} = \frac{z}{x-u}. \tag{3.19}$$

If we then substitute D with the right-hand side of (3.18) and rearrange, we obtain:

$$[n^2(d^2+u^2)-u^2](x-u)^2 - u^2z^2 = 0. \tag{3.20}$$

Note that in (3.18) D is a function of pixel location, u, so rays no longer converge to a single point. As a result, a single focal length adjustment [12] cannot fully compensate for refractive distortion. Also, the presence of z in

Algorithm 1 Projecting a global space point, $\mathbf{M}$, into image space, supposing that the viewpoint $\mathbf{V}$ is at the origin.

```
1: Compute $\mathbf{M}'$, the local space representation of $\mathbf{M}$, using (3.6)
2: Compute $z = \| \mathbf{M}'_{\text{proj}} \| - d$, the length of the projection of $\mathbf{M}'$ onto the refractive interface's normal, minus the distance from $V$ to the interface
3: Compute $x = \| \mathbf{M}'_{\text{proj}} - \mathbf{M}' \|$, the perpendicular distance between the refractive interface's normal and $\mathbf{M}$
4: Solve (3.20) for roots $u_i, i \in \{1, 2, 3, 4\}$
5: Find $u = u_i$ such that $u_i \in [0, \mathbf{M}_y]$ (only one such root exists [15]).
6: Let $\hat{\mathbf{d}}$ be a unit vector in the same direction as $\mathbf{Q}' - \mathbf{C}$, where $\mathbf{Q}'$ is the intersection point between the line $\overline{\mathbf{VM}}$ and the refractive interface
7: Compute $\mathbf{Q} = \mathbf{O} + u\hat{\mathbf{d}}$,
8: Project $\mathbf{Q}$ into the image, i.e., $K\mathbf{Q}$ where $K$ is the camera's intrinsic matrix
```

(3.20) implies that u depends on the 3D location of $\mathbf{M}$, hence applying radial distortion correction will not work because radial distortion can only compensate for image-space distortions [46].

Equation (3.20) is a fourth degree polynomial in u. Such a polynomial has four roots, but only one root is meaningful. In particular, the root corresponding to a $\mathbf{Q}$ whose y-coordinate lies in the interval $[0, \mathbf{M}_y]$ is physically meaningful [15]. Once we have projected $\mathbf{M}$ onto the refractive interface to obtain $\mathbf{Q}$, we can project $\mathbf{Q}$ into the camera to find the actual image point for $\mathbf{M}$. Similar derivations can be found in other works [9, 12, 15, 46]. Algorithm 1 describes in detail the process of projecting a point into an image under the influence of refraction and Appendix B provides the corresponding C++ code.

Figure 3.9 shows examples of the refractive epipolar curve using (3.20). Four features (center of the circled regions) have been selected in the left view. Zoomed-in regions of the highlighted boxes in the right view show the corresponding epipolar lines and curves. As one can see, when ignoring refraction, the deviation from the selected feature is significant, up to 10 pixels in this example. The deviation is severe enough to make stereo matching a difficult problem when using any approach that ignores the effects of refraction.

Also of interest is the impact of the camera's focal length on the epipolar

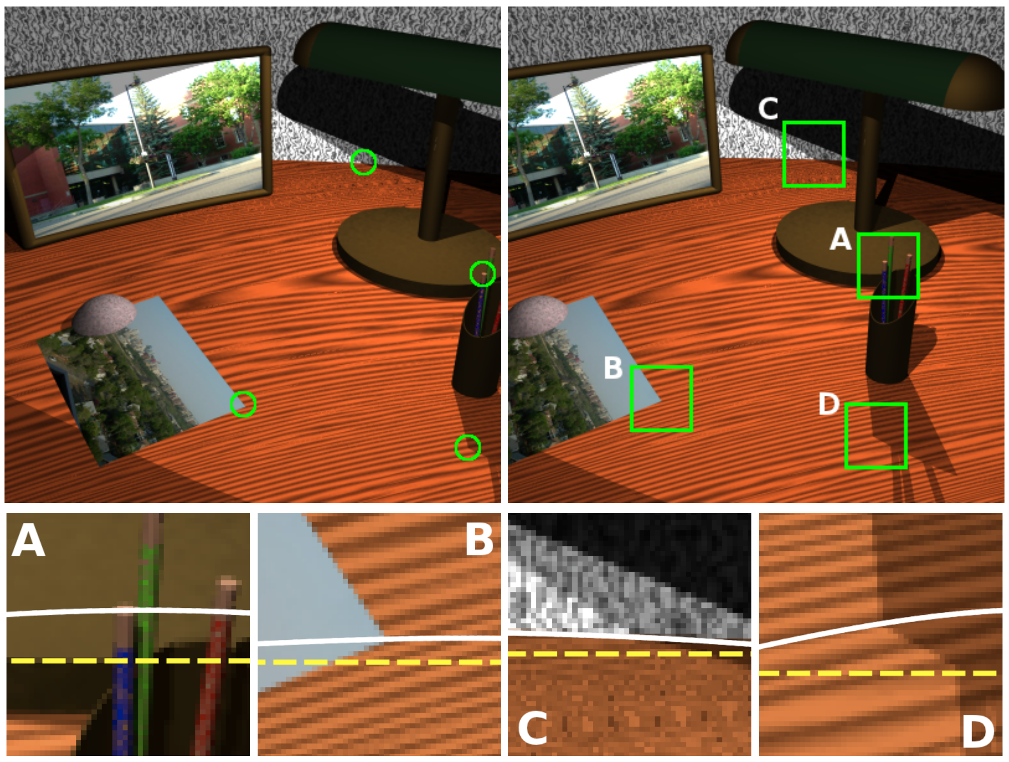


Figure 3.9: Selected features in the left view (top-left) with their corresponding epipolar lines (dashed line, bottom) and curves (solid line, bottom) in the right view (top-right) of a synthetic scene. Note that the refractive interface is not parallel with the imaging plane. Without properly compensating for refraction, the corresponding epipolar line misses the feature by up to 10 pixels.

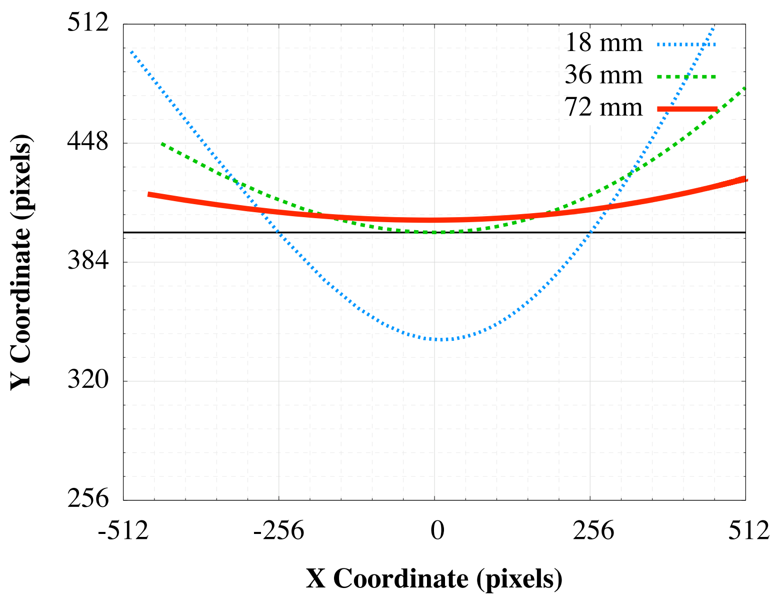


Figure 3.10: Effect of a focal length adjustment on the epipolar geometry. The black horizontal line represents the standard epipolar line when no refractive interface exists. Given a camera at a fixed location and a refractive interface which is at a fixed distance from the camera and is coplanar with the camera's imaging plane, we can see how widening the field of view (i.e., decreasing the focal length) warps the epipolar geometry significantly.

geometry. In particular, wider angle lenses permit light rays with larger angles. The larger the angle, the greater the amount of bending imposed by the refractive interface. Figure 3.10 shows the significance of a focal length adjustment given every other part of the system remains fixed. As we increase the focal length, which in this case decreases the field-of-view, the epipolar curve starts to straighten. This observation can be described using (3.18). In particular, suppose the focal length of the camera and d are the same. As d increases, the first term under the square root begins to dominate the second, and eventually (3.18) becomes approximately nd. Since D is now fixed at nd, we essentially have a single viewpoint camera, which in turn means our epipolar geometry will be lines and not curves.

Chapter 4

Underwater Stereo

In this chapter, we introduce our adaptation of the previous chapter's theory to develop a method for underwater stereo matching. We begin by introducing our method for calibrating the parameters of the refractive interface, followed by our incorporation of epipolar curve sampling into a stereo matching framework.

4.1 Refractive Interface Calibration

4.1.1 Method

Figure 4.1 shows a general setup for underwater stereo matching. Before the various equations in Section 3.4 can be used to project 3D points in water onto the imaging planes of individual cameras, we must first know the parameters which define the refractive interface for each camera. In our system we model a plane P as:

$$P : \vec{\mathbf{n}} \cdot \mathbf{x} = d. \tag{4.1}$$

This model has three free parameters: two specifying the normal of the plane and one specifying the distance d from the origin to the plane (in the direction of the normal). In particular, we choose a pixel $p_{\mathbf{n}}$ on the imaging plane to define the normal. The direction of a ray through this pixel will be equal to the normal of the plane:

$$\vec{\mathbf{n}} = K^{-1} p_{\mathbf{n}}. \tag{4.2}$$

Before calibrating the interface parameters we need to have our cameras calibrated in air so that we know their intrinsic and extrinsic parameters. This allows us to map rays and points to and from global space to each of the cameras' local coordinate systems. One reason for separating these two calibrations

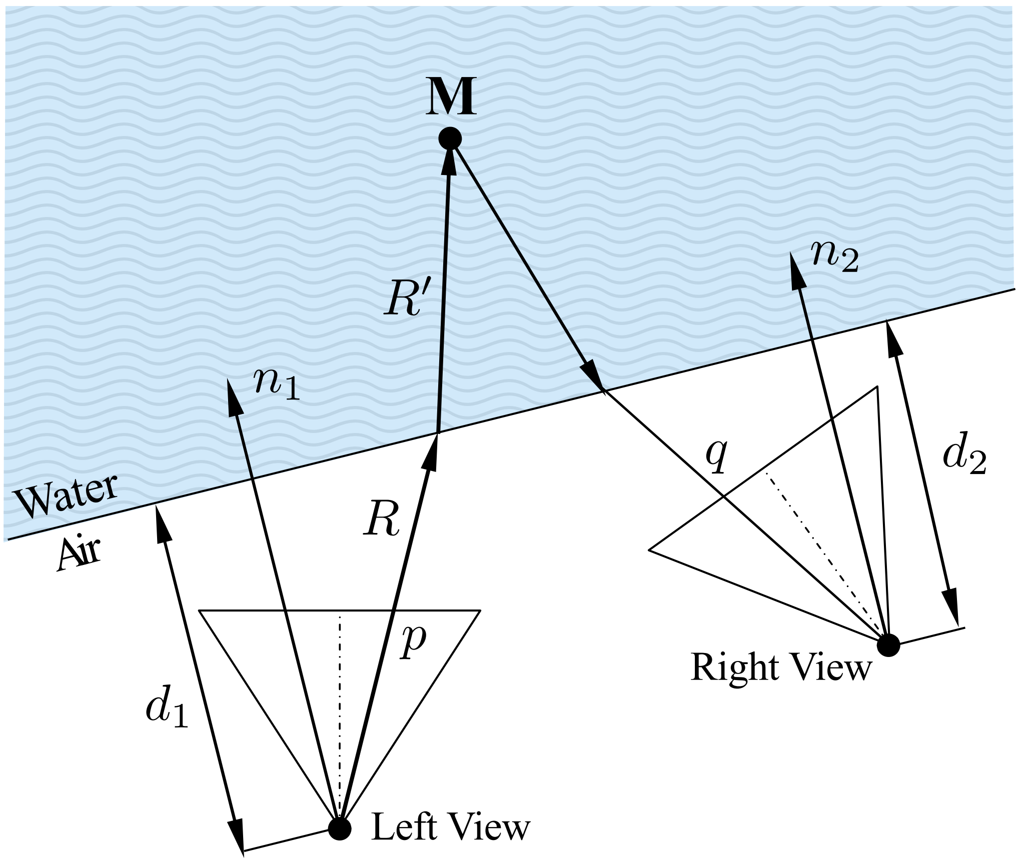


Figure 4.1: Refractive stereo setup.

is to avoid accidentally absorbing any refractive distortion into the lens distortion parameters [30]. Also, by separating these two calibrations we reduce the number of unknowns in both phases, keeping the complex nonlinear structure of the refractive calibration separate.

Our calibration method performs a nonlinear optimization over a set of feature correspondences in two views. Let $f = \{p, q\}$ be a feature correspondence between two views, $r(p; P, n)$ be the ray casted through pixel p and refracted against refractive interface P with a refractive index ratio of n, and $C(r_1, r_2)$ be the smallest distance between rays r_1 and r_2 (see Appendix D). For a given estimate of the parameters we define the error of feature correspondence f as

$$e(f; P_1, P_2) = C(r(p; P_1, n),\ r(q; P_2, n)), \tag{4.3}$$

or rather, the distance between the refracted rays of the features in a given correspondence. To be as general as possible, each camera is modelled with its own refractive interface, $P_1 = \{\vec{\mathbf{n}}_1, d_1\}$ and $P_2 = \{\vec{\mathbf{n}}_2, d_2\}$. This can, for example, handle situations when the cameras are in their own housings (Figure 1.1), or

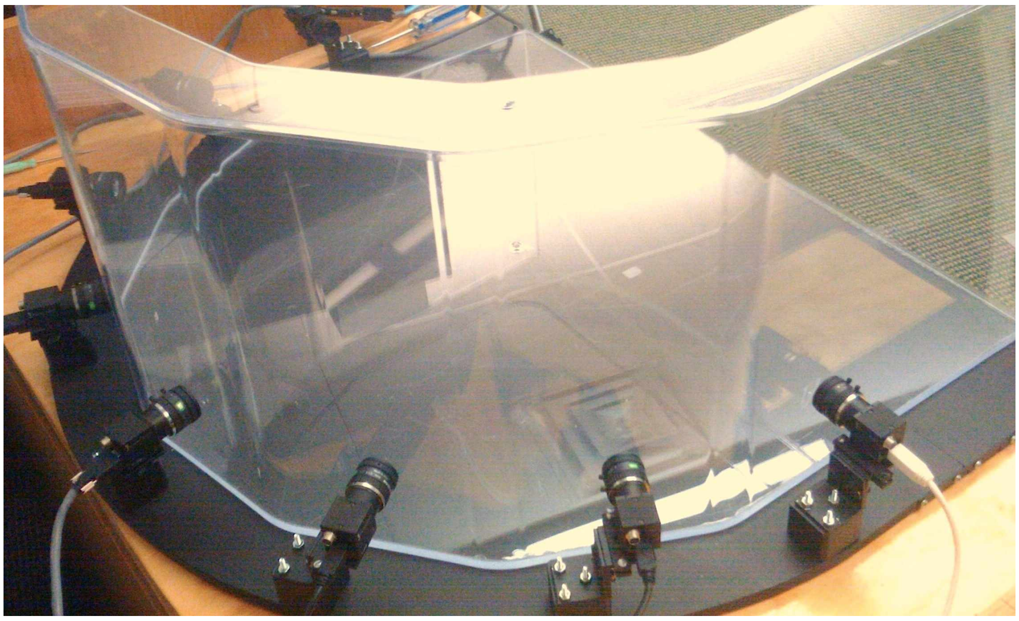


Figure 4.2: Our tank array has a different side for each camera. Since cameras do not share the same refractive interface, we need a model which takes into account that each camera is behind a different refractive interface.

when viewing an underwater scene from different sides of a tank (Figure 4.2). If both cameras share the same refractive interface then we can reduce the model to a single plane (i.e., three parameters) and use the extrinsic parameters of the cameras to map this plane into the local space of each camera.

We choose the distance between two rays for computational efficiency, since an image-based error metric will require solving the quartic in (3.20). Suppose τ_i is the perpendicular distance from the midpoint of r_1 and r_2 to image i. By scaling the distance between the two rays by this value we can approximate an image space distance metric. The total error over all feature correspondences is then defined as:

$$E = \sum_f \left(\frac{e(f; P_1, P_2)}{2\tau_1} + \frac{e(f; P_1, P_2)}{2\tau_2} \right). \tag{4.4}$$

Pseudocode for computing (4.4) is given in Algorithm 2. We can find the parameters of P_1 and P_2 by minimizing E with any nonlinear optimization algorithm. We use the Levenberg–Marquardt algorithm [35], with finite differences for approximating function gradients. The initial parameters are set such that the normal of the refractive planes are parallel to the optical axis of their corresponding

Algorithm 2 Computation of E in (4.4)

```
1:  Let $E = 0$
2:  for all feature correspondences $f = p \leftrightarrow q$ do
3:      Cast a ray $r_1$ through $p$ and ray $r_2$ through $q$
4:      Find refracted ray $r'_1$ using the refractive interface $P_1$
5:      Find refracted ray $r'_2$ using the refractive interface $P_2$
6:      Let $dist = C(r'_1, r'_2)$, the distance between rays
7:      Compute midpoint $m$ between rays $r'_1$ and $r'_2$ (Appendix D)
8:      Compute $\tau_1$, perpendicular distance of $m$ from left image
9:      Compute $\tau_2$, perpendicular distance of $m$ from right image
10:     Let $E = E + \frac{dist}{2\tau_1} + \frac{dist}{2\tau_2}$
11: end for
```

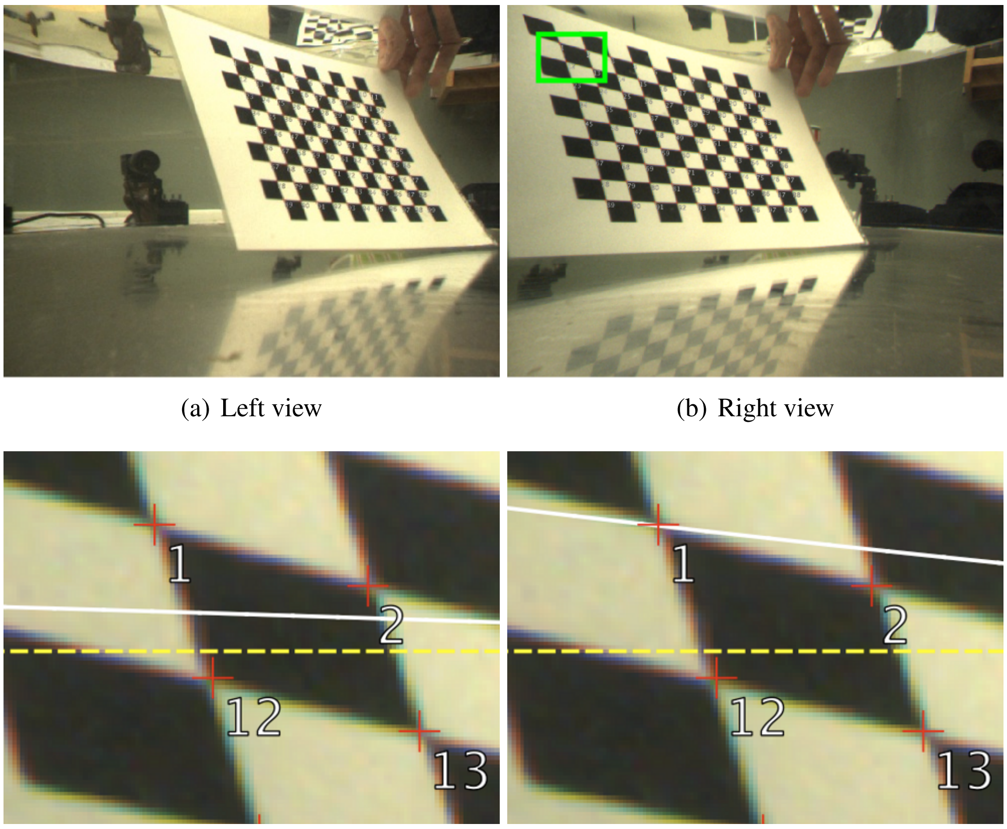
camera. That is,

$$\mathbf{n}_i = K^{-1} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}, \tag{4.5}$$

where x_0, y_0 are the pixel coordinates of the principal point, as defined in Section 3.1.2. The distance to the interface is set to some fixed value, as selected by the user.

Figure 4.3 illustrates the impact of the calibration process. To simplify the detection of corresponding features, a planar checkerboard pattern is used in our experiment. However, our calibration approach does not assume the corresponding features are on the same plane, so any feature can be used. The two stereo views are accurately rectified in the air so that any remaining distortion is caused by the refractive interface only. The results clearly show the importance and effectiveness of the proposed refractive interface calibration.

We have found that (4.4) is quite sensitive to the initial parameter estimates, with the interface distance parameters (i.e., d_1 and d_2) being the most unstable. Of great importance is capturing data as close as possible to the refractive interface. Such data will act as a constraint on the interface distance parameter, preventing it from being too large or too small. In our own experiments we found that the impact of this data is significant. In one scenario we placed our stereo camera very close to the interface, which prevented us from capturing



(a) Left view (b) Right view

(c) Before calibration (d) After calibration

Figure 4.3: The epipolar curve (solid line) and epipolar line when ignoring refraction (dashed line) for checkerboard corner feature #1 before and after calibration. Before calibration the interface parameters are initialized such that the normal of the interface is the same as the camera's optical axis and its distance is a fixed value. Images (c) and (d) are zoom-ins of the highlighted area from (b) and show the impact of a properly calibrated interface.

data with the checkerboard pattern near the interface. In this situation, we found that the initial estimates for d_1 and d_2 had to be within a couple of centimetres of the measured value so that the optimization routine would converge. In another scenario, we placed the stereo camera further away so that we could place the checkerboard pattern directly on the interface. In this scenario, the initial estimates for d_1 and d_2 could be up to two metres off of the measured value. One of the negative aspects of our calibration routine is that it is difficult to establish feature correspondence between cameras which are very close to the interface.

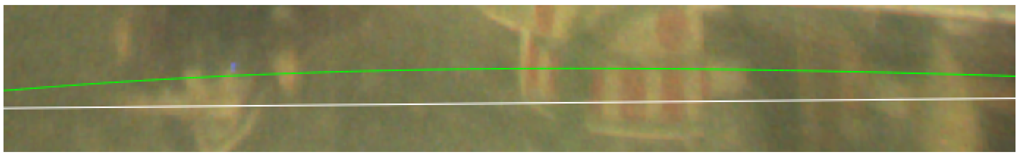
4.2 Stereo Matching

4.2.1 Method

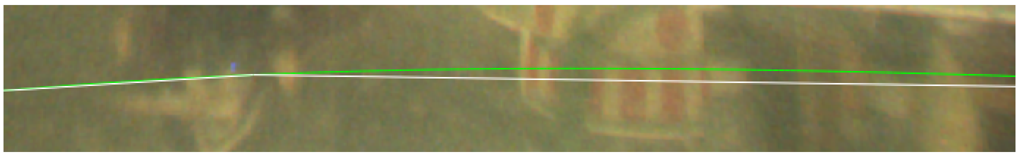
The various equations in Section 3.4 allow us to compensate for the effects of refraction when performing stereo matching. The most important part of our approach is how to sample the epipolar curve. First, a ray is casted through a pixel and refracted against that view's refractive interface. Multiple samples from this ray are projected into the other view, using Algorithm 1.

This process could lead to both oversampling and undersampling. Oversampling occurs when multiple samples project to the same pixel. In this case we have many samples that would share a similar correlation score, since windows would be centered on the same pixel. Instead of considering samples individually, we discard projected samples until we find two samples that are at least a pixel apart. Undersampling occurs when projected samples are more than a pixel apart. In this case we could potentially miss the true correlation peak, so we sample all pixels along the line joining two samples. Figure 4.4 shows the impact of the number of samples on the shape of the epipolar curve. In our own work we have found that 25 or more samples sufficiently describes this epipolar curve in the image.

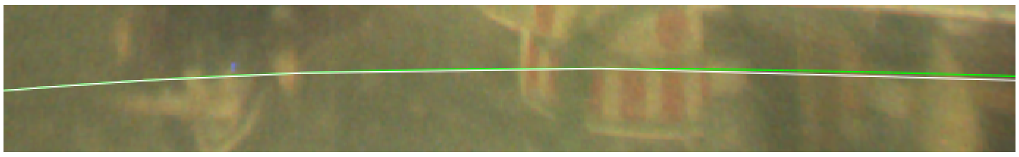
As far as we can tell, no previous approach has incorporated (3.20) into a stereo matching framework. Algorithm 3 details our approach to estimating the epipolar curve imposed by a refractive interface. For a local stereo method, we can simply compare each pixel $q' \in \mathcal{L}$ to p, for example, using normalized-cross correlation (NCC) over a small window. We then take the best result as our hypothesis for pixel p, i.e., solve (2.3).



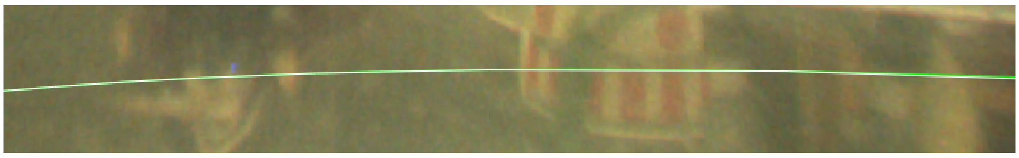
(a) $k = 2$



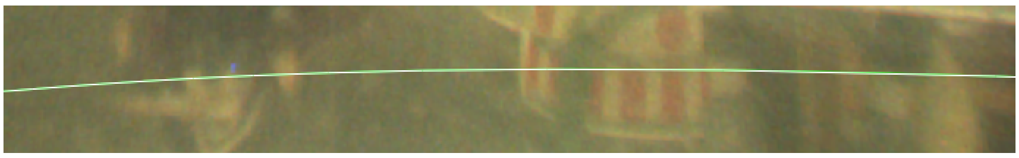
(b) $k = 5$



(c) $k = 10$



(d) $k = 15$



(e) $k = 25$

Figure 4.4: The impact of k in Algorithm 3. The sampled curve (white) is superimposed over a densely sampled curve (green) where $k = 1000$. We have found that $k = 25$ is sufficient for accurately sampling the epipolar curve in a typical setup.

Algorithm 3 Piecewise-linear sampling of the epipolar curve $\mathcal{L}$ for pixel p in an image (Figure 4.1).

```
1: Find the ray $R$ passing through pixel $p$ using (4.2).
2: Refract $R$ at the camera's local planar interface to get $R'$.
3: Intersect $R'$ with $k$ depth planes, giving $k$ sampled points $\mathbf{M}_i$.
4: for all $1 \leq i \leq k$ do
5:     Find $q_i$, the projection of $\mathbf{M}_i$ into the other image using Algorithm 1
6: end for
7: $\mathcal{L} = \bigcup_{2 \leq i \leq k} \{\overline{q_{i-1}q_i}\}$
```

In this thesis, we implement the method by Campbell et al. [7] as described in Section 2.2.5. We simply substitute the epipolar line in their method with our epipolar curve. Our implementation of this method uses the tree-reweighted message passing method [24,51] provided in the Middlebury MRF optimization framework [43]. We also use geodesic support weights [21] for weighting pixels within windows (Appendix E).

Chapter 5

Experiments

In this section we outline some of our binocular and multi-view stereo experiments. Unless otherwise stated, for all of our experiments we use the method described in Section 4.2 with parameters $K = 9$, $N = 3$, $\beta = \lambda = 1$, $\phi_{\mathrm{U}} = 0.5$, $\psi_{\mathrm{U}} = 0.002$, and a 5×5 window for NCC. We set $\alpha = 1$ cm in (2.14) so that a depth hypothesis in one view needs to be within one centimetre of its corresponding depth hypothesis in another view. Finally, all depth maps color depth values linearly, with black being the closest and white being the farthest.

5.1 Binocular Stereo

Our binocular experiments use the Bumblebee (BBCOL-40) from Point Grey Research, imaging several figures and objects placed in a water-filled tank[1]. The stereo matching methods we compare include:

- **Focal length correction**; The only adjustment made is to the focal length, which is scaled by the refractive index ratio so as to produce depth values that are more physically accurate [28].

- **Radial distortion correction**; a series of images are taken of a checkerboard calibration pattern submerged in water. The cameras are then calibrated using a traditional calibration technique [57], with the refractive distortion being “absorbed” into the radial lens distortion parameters.

- **Refractive correction**; our own approach, as described in Section 4.2. Cameras are initially calibrated in air using a traditional calibration tech-

[1] To focus on just the objects and remove any background noise, we use binary masks for all experiments.

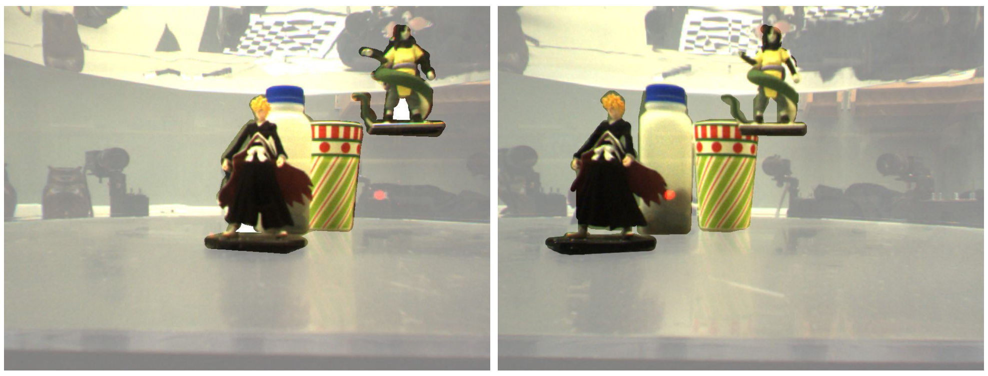
nique [57]. Also, the refractive interface parameters are calibrated using a set of images of checkerboard calibration patterns submerged in water, as described in Section 4.1.

Figures 5.1, 5.3, and 5.5 show results from our two experiments. The setup for Figure 5.1 involved placing the camera approximately 10 centimetres from the tank, with the imaging plane approximately parallel with the side of the tank. In figures 5.3 and 5.5 we placed the camera at an angle of approximately 20° with the tank. Our experiments consistently retain more depth hypotheses after cross-checking.

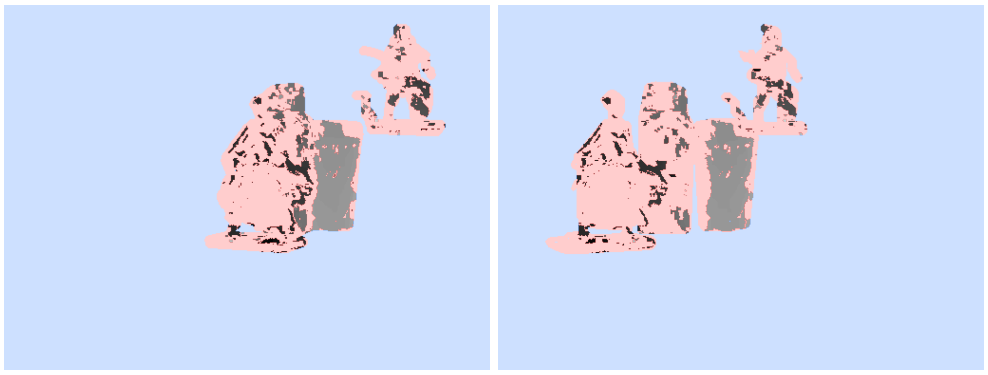
In both experiments we obtain visibly better results, with larger "blobs" of consistent depth hypotheses. In particular, the raised figures are far more distinctive in our depth maps. Since they are found closer to the boundaries of the images, more refractive distortion exists. The other two approaches cannot find the correct matches since the distortion is too severe.

Figures 5.2, 5.4, and 5.6 show the percentage of pixels with depth hypotheses, verifying the visibly better results. In all cases we obtain roughly the same number of depth hypotheses before cross-checking, but approximately 5-40% more depth-hypotheses remain after cross-checking compared to focal length correction, and 10-20% more depth-hypotheses remain after cross-checking compared to radial distortion correction. Since the second experiment has a non-parallel interface, more refractive distortion is present and hence we see that the margin between our method and the other methods is even greater than that of the first experiment, which had a parallel interface.

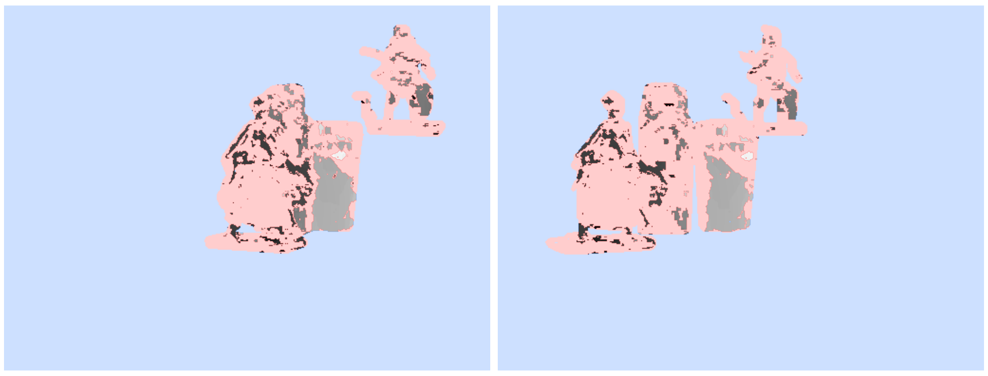
It is important to note several reasons why the obtained depth hypotheses are not any denser. First, the data itself is challenging due to a lack of texture (e.g., figure with a black cloak). Also, refractive distortion is most significant in only a small portion of the image. Regardless, there is still a noticeable improvement. Figure 5.7 also shows how it is important to remember the difference between consistency and correctness when reading figures 5.2, 5.6, and 5.4. In particular, focal length correction and radial distortion correction both have roughly the same percentage of consistent depth hypotheses, yet the raised figure is fare more distinguishable with radial distortion corection.



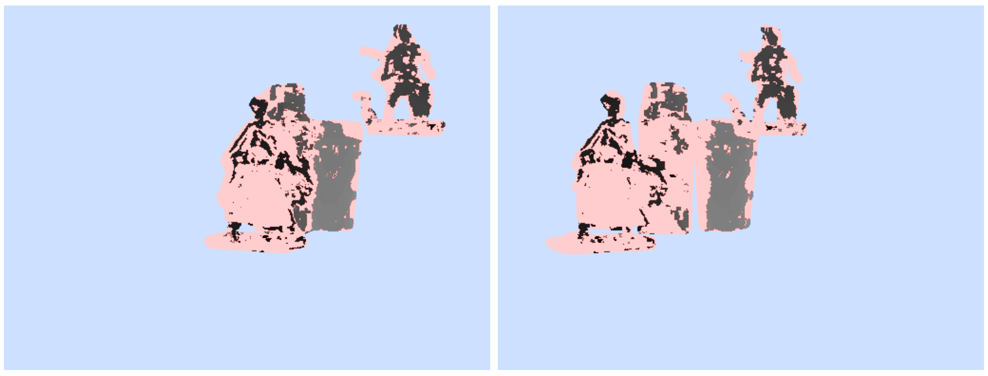
(a) Original Images



(b) Focal Length Correction



(c) Radial Distortion Correction



(d) Refractive Correction

Figure 5.1: Depth maps produced from binocular stereo experiment #1. Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink.

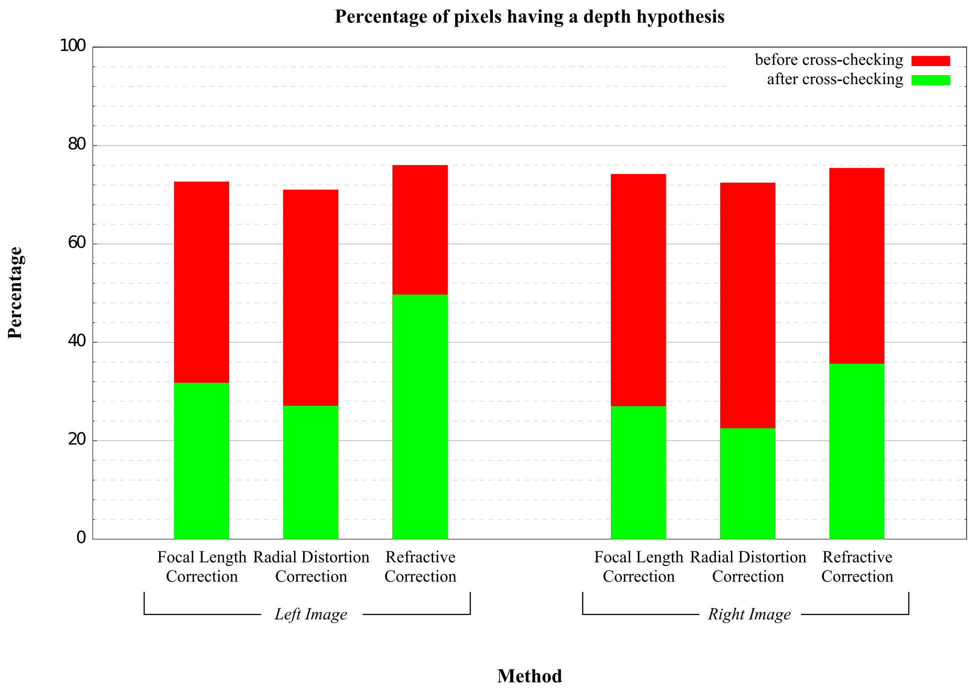
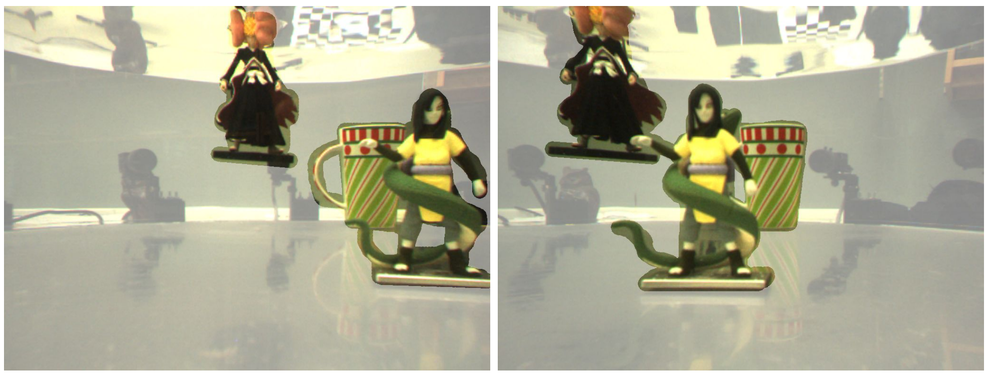
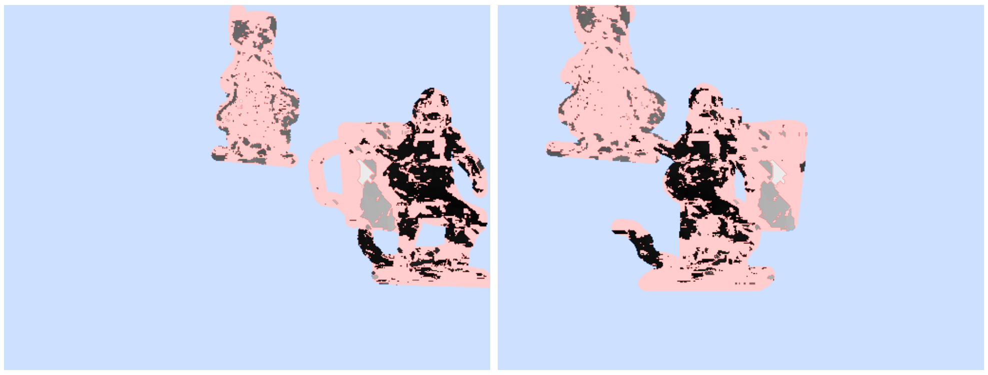


Figure 5.2: Percentage of pixels with a depth hypothesis before and after cross-checking for binocular stereo experiment #1 (Figure 5.1).

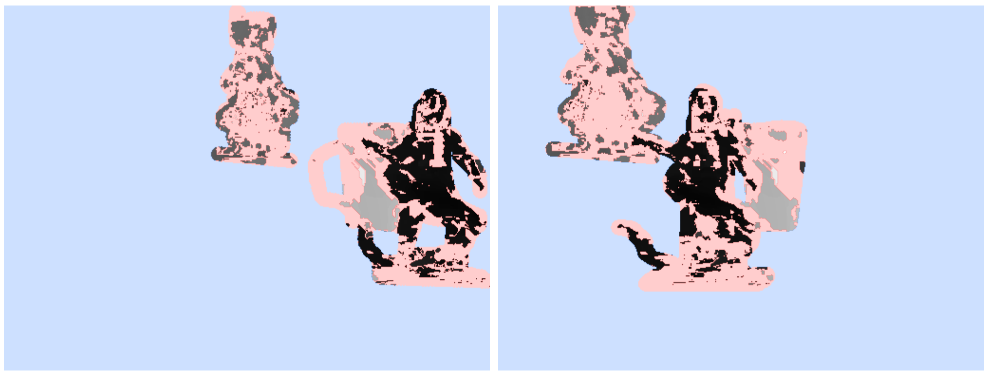
Also, of particular importance is the calibration of the interface, and its impact on the matching process (Figure 5.8). The calibrated normals differ by only a few pixels – which is equivalent to a shift of less than a degree – and the interface distance parameters differ by about 1.5 millimetres. Nevertheless, a significant increase in depth hypotheses can be seen.



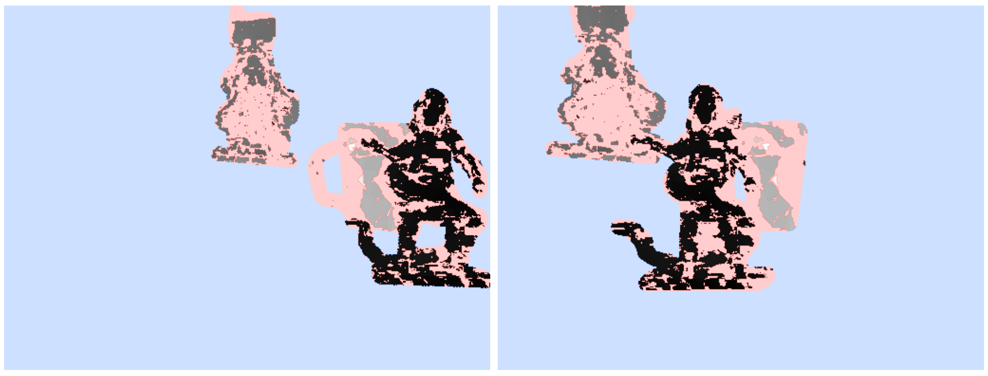
(a) Original Images



(b) Focal Length Correction



(c) Radial Distortion Correction



(d) Refractive Correction

Figure 5.3: Depth maps produced from binocular stereo experiment #2. Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink.

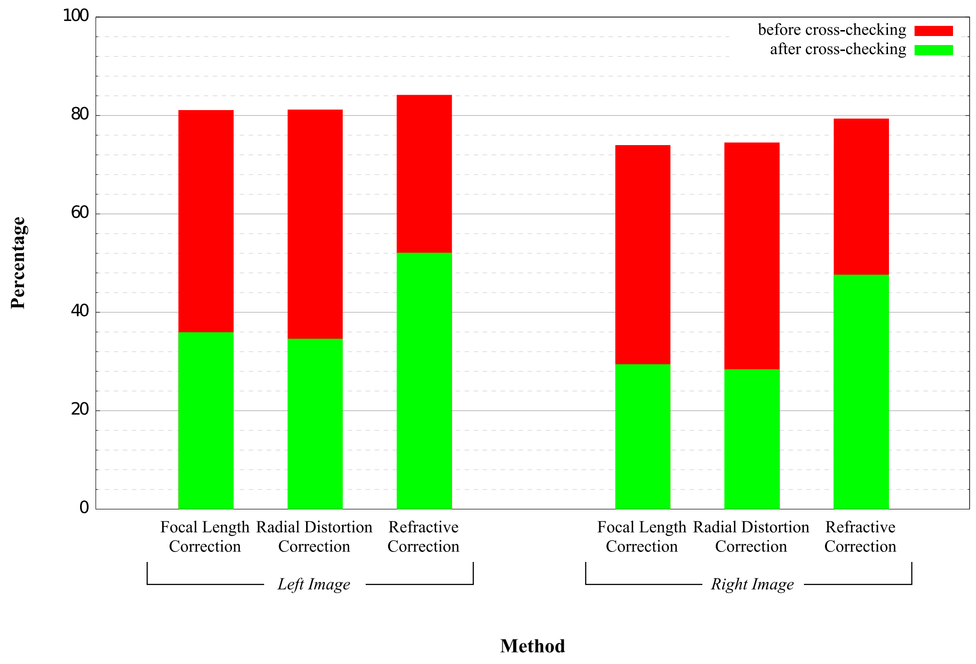


Figure 5.4: Percentage of pixels with a depth hypothesis before and after cross-checking for binocular stereo experiment #2 (Figure 5.3).

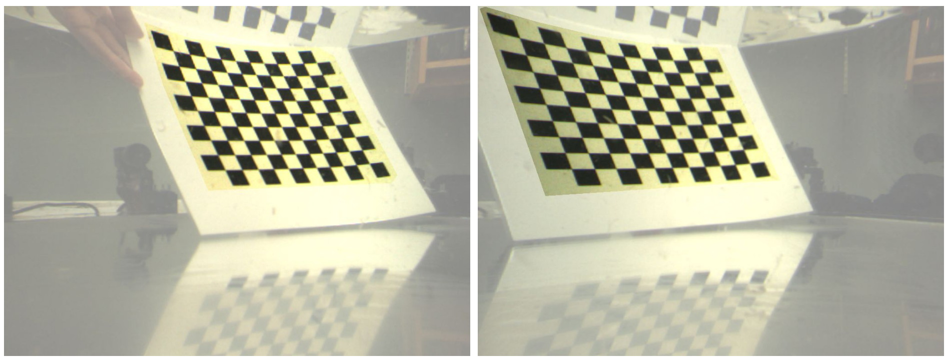
5.2 Multi-View

Our second set of experiments target multi-view stereo with synthetic data. One of the major benefits in using synthetic data sets is to show how well a method behaves in the absence of any external sources of error. In the case of stereo, real data can introduce additional error from, for example, calibration error. Another benefit to synthetic data is that a perfect ground truth can be used to establish error both quantitatively and qualitatively.

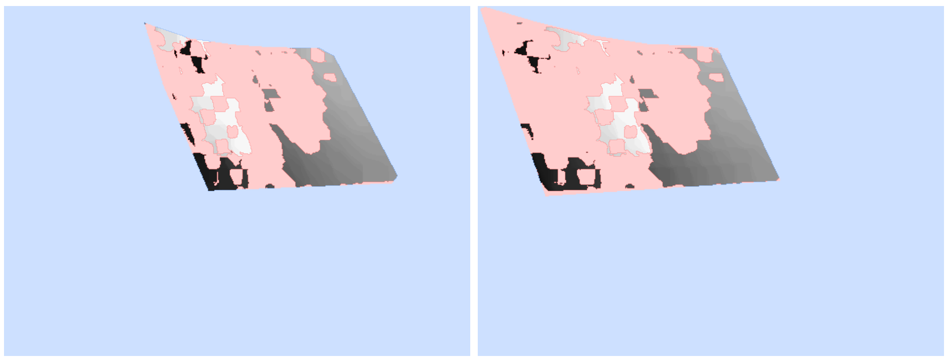
For our synthetic data, 16 cameras were arranged uniformly along a unit semi-circle ((Figure 5.9)). A model of the Stanford Bunny was placed near the center and textured to allow for good correlation. An invisible refractive interface was placed in front of the cameras in various configurations. POV-Ray [3], a well-established ray tracing program, was then used to render the scene.

Figure 5.10 shows the result when we introduce a refractive interface that is parallel to every camera's interface at a distance of 0.01 units[2] (Figure 5.9(a)). Two key observations can be made from this data set. First, our own method

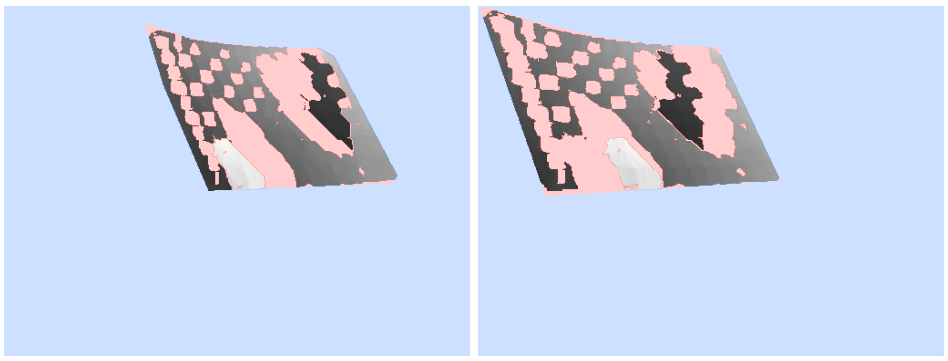
[2] Synthetic data sets were not rendered in a basis which has a physically meaningful unit, hence we simply use the term "unit".



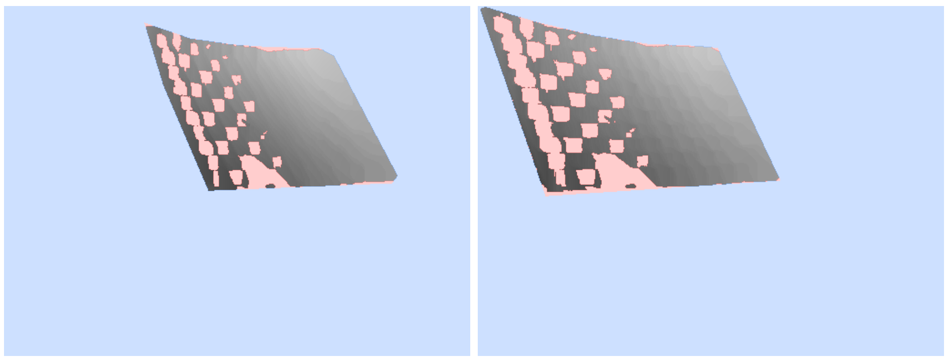
(a) Original Images



(b) Focal Length Correction



(c) Radial Distortion Correction



(d) Refractive Correction

Figure 5.5: Depth maps produced from binocular stereo experiment #3. Window size was increased to 31×31 to better avoid matching failure due to the lack of texture and repeated pattern of the checkerboard. Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink.

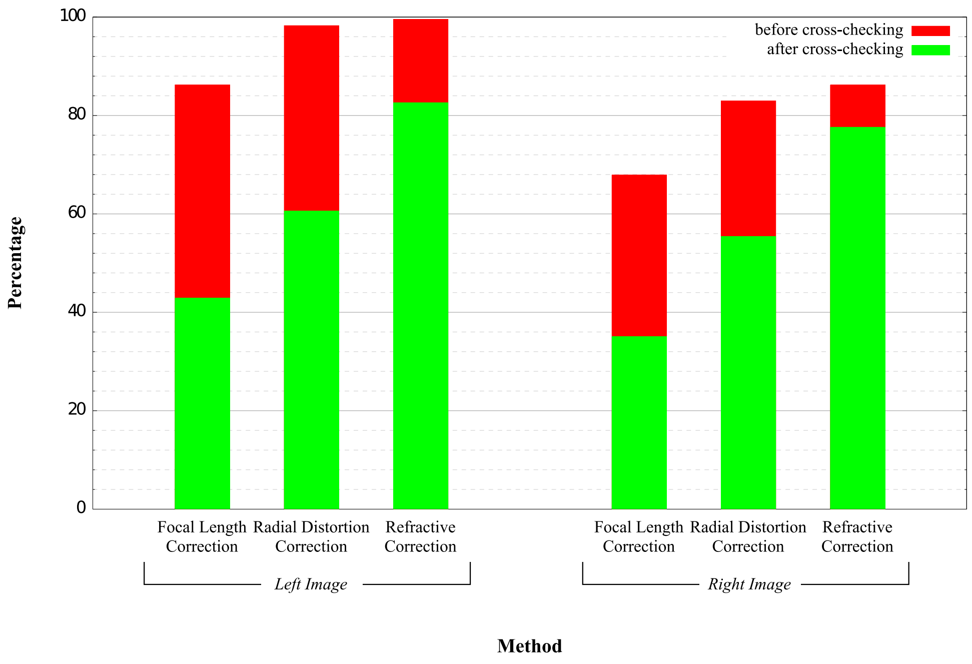
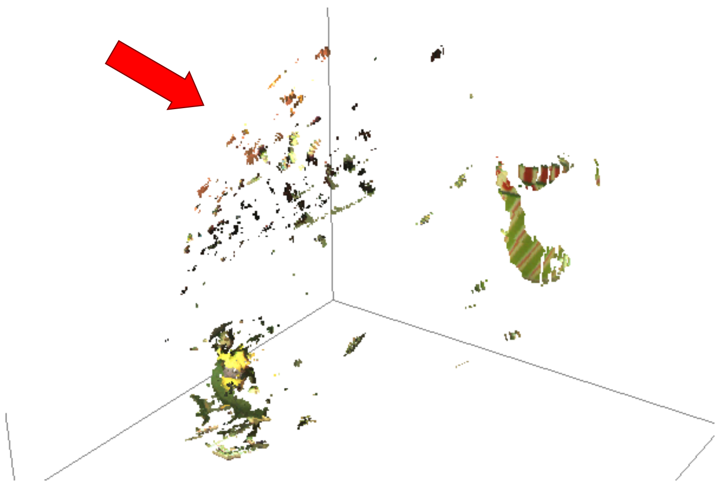


Figure 5.6: Percentage of pixels with a depth hypothesis before and after cross-checking for binocular stereo experiment #3 (Figure 5.5).

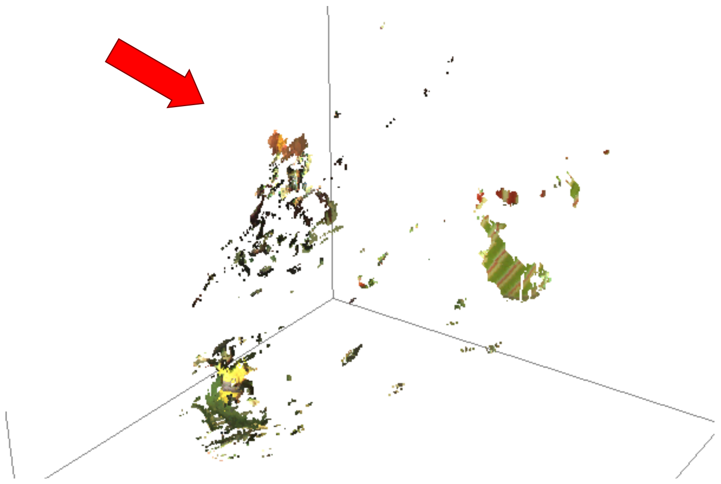
(Figure 5.10(e)) produces far more depth hypotheses than focal length correction (Figure 5.10(c)) and roughly the same amount as radial distortion correction (Figure 5.10(d)). Second, since the interfaces are parallel to the imaging planes, refractive distortion is most significant towards the boundaries, where we see less depth depth hypotheses when using just focal length correction. Finally, since most views of this scene fell into a relatively small amount of depths, radial distortion correction provided good results (Figure 5.10(d)).

Our second experiment incorporates refractive interfaces that are not parallel to the imaging planes of each camera. In addition, even- and odd-numbered views have interface parameters that differ from each other. These two factors incorporate significant refractive distortion into various areas of the images (Figure 5.9(b)). Figure 5.11 shows just how significant the problem of refraction can be. Focal length correction provides poor results, to the point that a bunny is barely recognizable (Figure 5.11(c)). Radial distortion correction provides significantly better results (Figure 5.11(d)), but our refractive correction still provides the best results (Figure 5.11(e)), comparable to that of Figure 5.10.

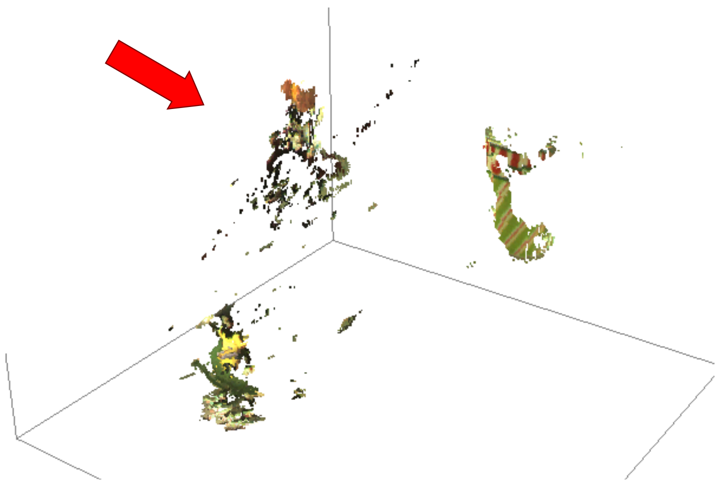
Figure 5.12 shows the root mean square (RMS) error values for all sixteen



(a) Focal Length Correction



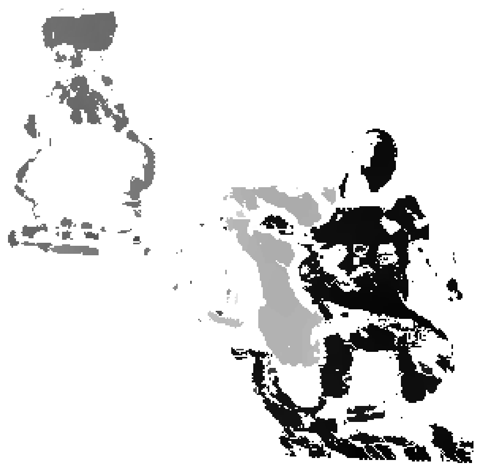
(b) Radial Distortion Correction



(c) Refractive Correction

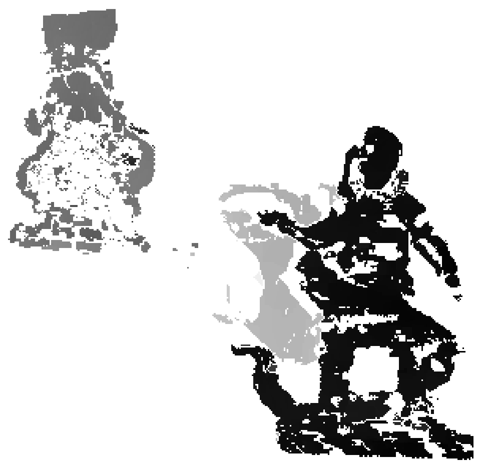
Figure 5.7: Three-dimensional renderings of depth maps produced for binocular stereo experiment #2. Focal length correction and radial distortion correction provide roughly the same number of depth hypotheses (figures 5.2 and 5.4) yet the lens distortion result is better. In particular, the shape of the raised figure (indicated by arrow) is more defined.

Estimated Parameter	**Left View**	**Right View**
Normal (p_x, p_y)	(746.57, 391.85)	(746.57, 391.85)
Interface Distance (m)	0.0435	0.0116



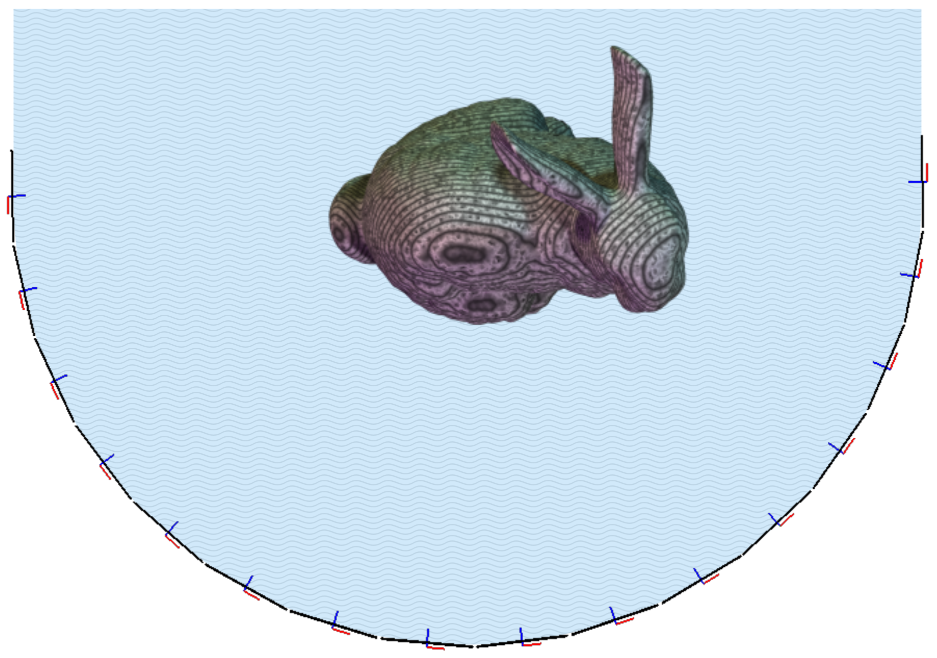
(a)

Estimated Parameter	**Left View**	**Right View**
Normal (p_x, p_y)	(741.51, 387.77)	(748.55, 388.26)
Interface Distance (m)	0.0422	0.0101

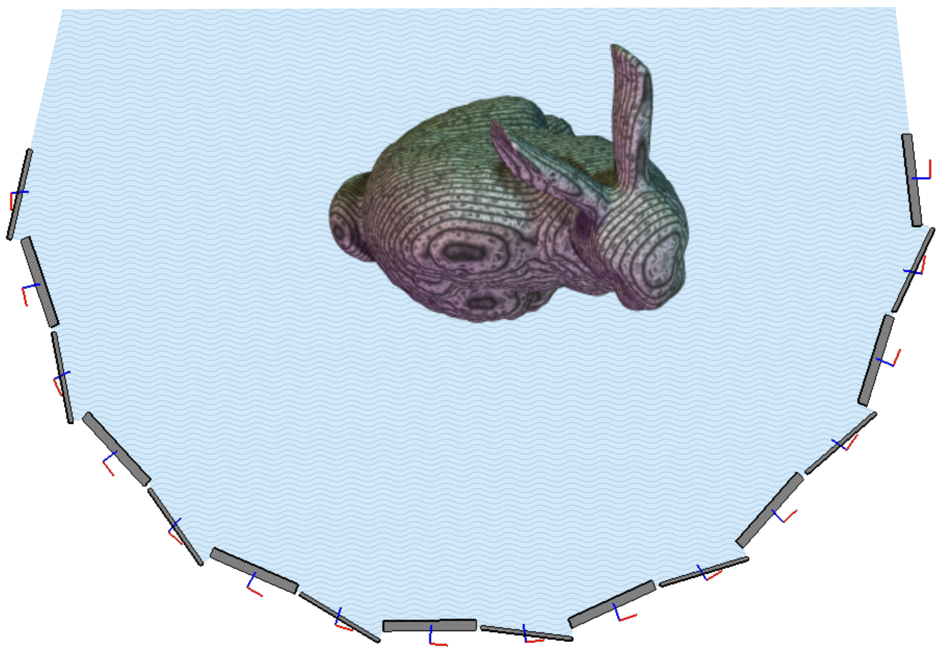


(b)

Figure 5.8: The impact of refractive interface calibration on stereo matching. Even though the calibrated interface in (a) is close to the calibrated interface in (b), a much larger set of depth hypotheses was obtained in (b). Pixel brightness corresponds to depth, with darker pixels being closer. Pixels with no depth hypothesis are white.



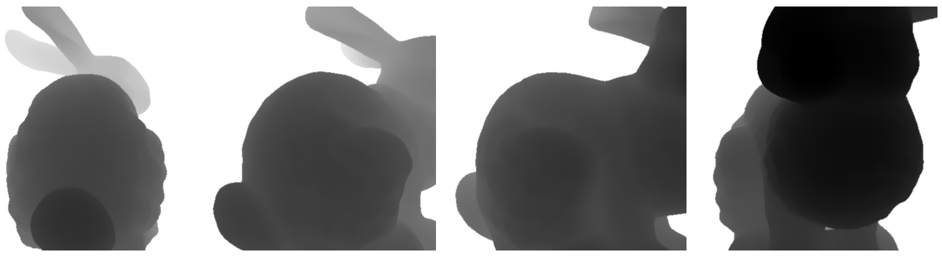
(a) Multi-view Experiment #1



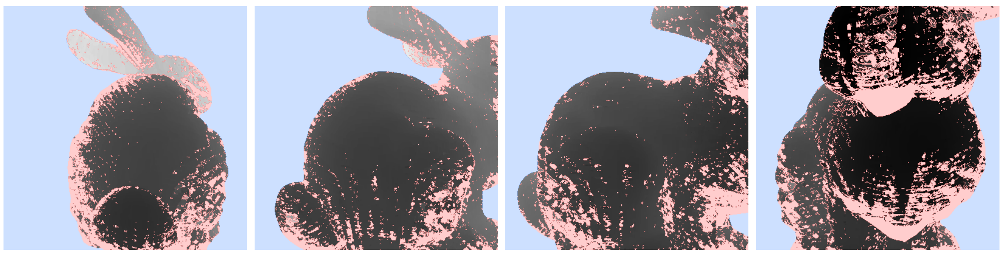
(b) Multi-view Experiment #2

Figure 5.9: Setup for multi-view experiments #1 and #2. Cameras are represented with their respective x-axes (red), optical axes (blue), and refractive interfaces (black).

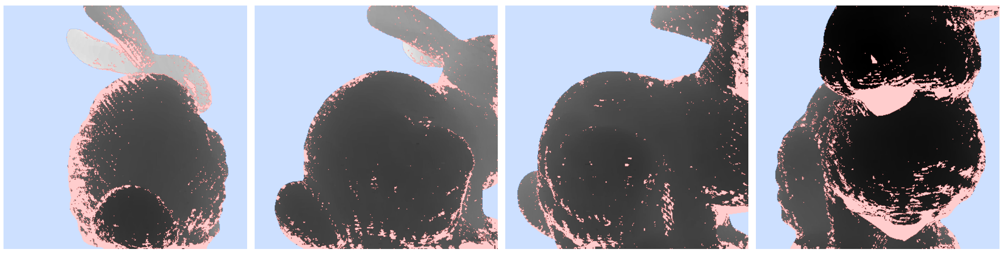
(a) Original Images



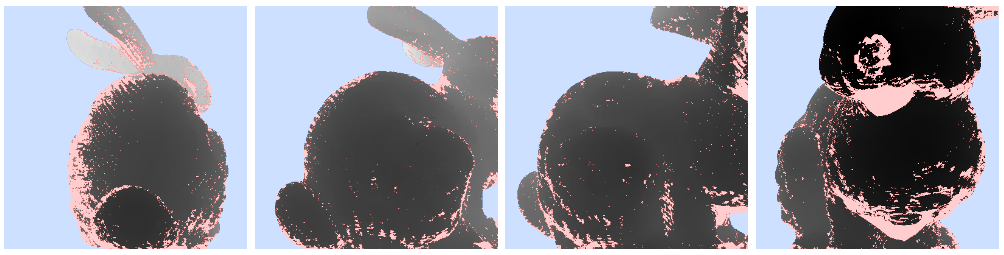
(b) Ground Truth Depth Map



(c) Focal Length Correction

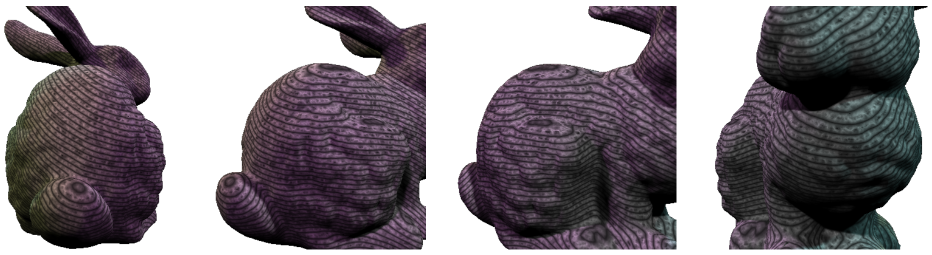


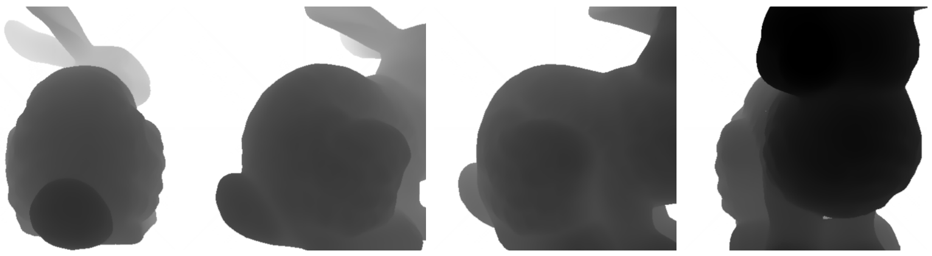
(d) Radial Distortion Correction

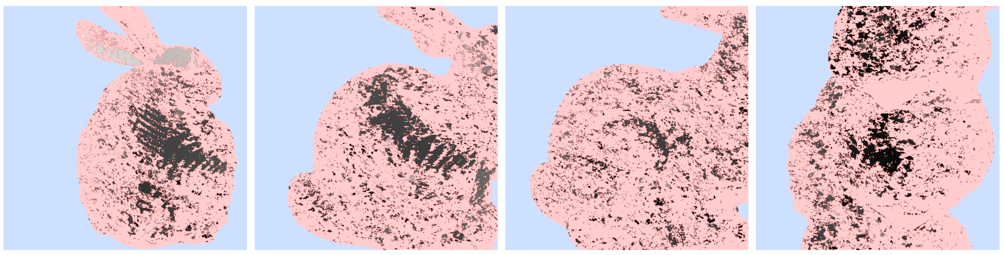


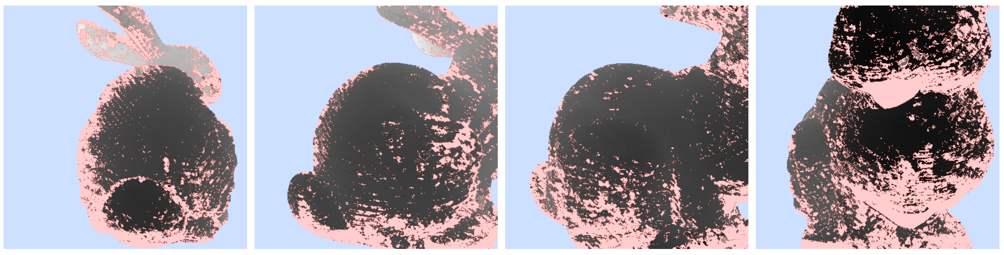
(e) Refractive Correction

Figure 5.10: Multi-view stereo experiment #1, using synthetic data. Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink. Since the interface is parallel, refractive distortion becomes more significant towards the boundaries of the image. Less depth hypotheses are found near the boundaries when only scaling the focal length, which ignores these distortions.


(a) Original Images


(b) Ground Truth Depth Map


(c) Focal Length Correction


(d) Radial Distortion Correction

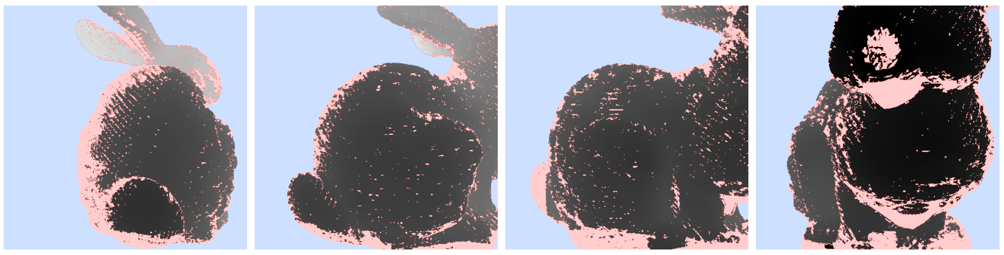

(e) Refractive Correction

Figure 5.11: Multi-view stereo experiment #2, using synthetic data. Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink. With non-parallel interfaces for each camera, refractive distortion is located throughout the images. Unless properly corrected for, the significant impact of refraction reduces good candidate matches.

views:

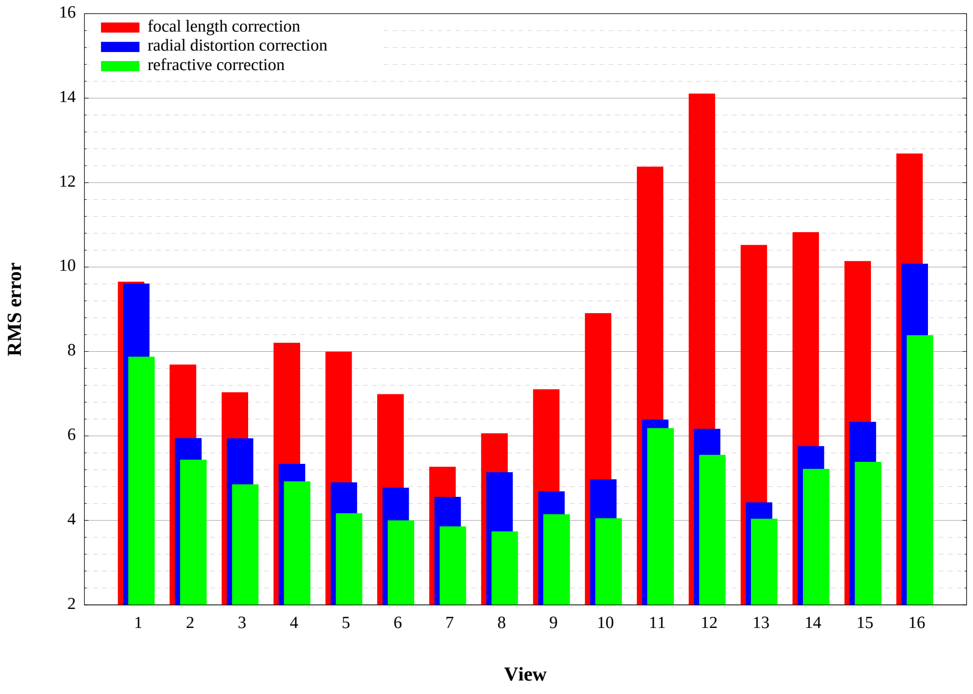
$$\mathrm{RMS}(\hat{\beta};\beta) = \sqrt{\frac{1}{n}\sum_{i}\left[\hat{\beta}_i - \beta_i\right]^2}, \tag{5.1}$$

where $\hat{beta}_i$ is the i^{th} observation and; β, the ground truth value for the i^{th} observation; and n, the total number of observations. In the case of computing RMS error values for depths maps, β are the ground truth depths, $\hat{\beta}$ the computed depths, and n the number of computed depth hypotheses. In Figure 5.12 we do not actually compare the difference in depth, but rather simply the difference in pixel intensity in the depth maps (which are directly proportional to the depth). As expected, our refractive correction consistently has a lower error than focal length and radial distortion correction. Also, the RMS error for refractive correction in both the parallel and non-parallel cases are essentially the same.

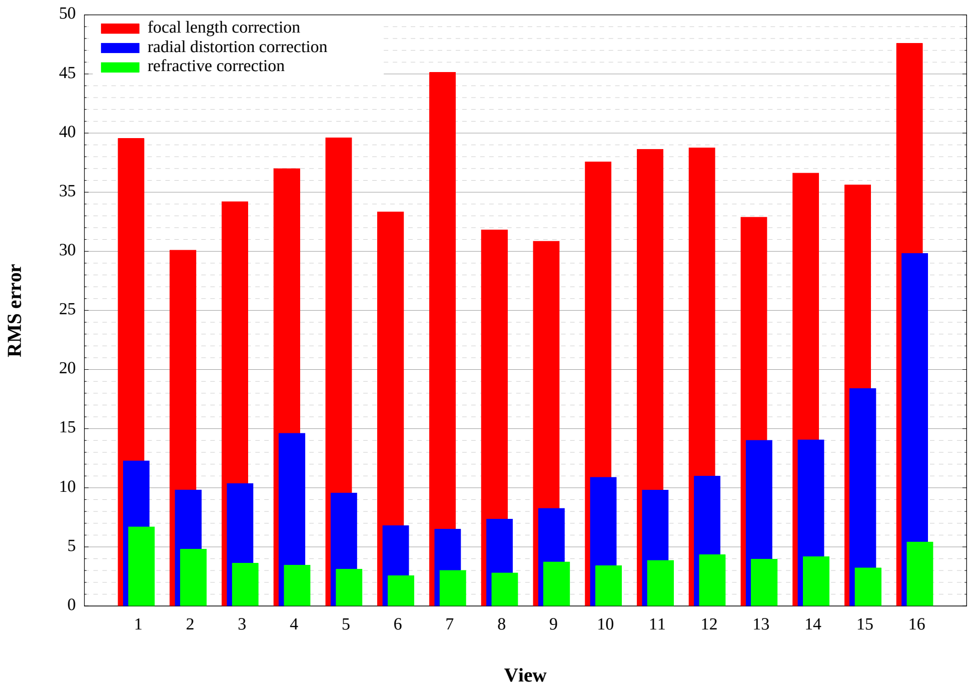
Even if other methods can obtain nice depth maps, ignoring refraction when reprojecting points will result in noise. In particular, reprojected points from two different views will not align properly. Figure 5.13 shows how the reprojection of these depth maps results in a noisy 3D point set. Our own method produces high quality results in both cases, whereas focal length correction (Figure 5.13(a)) and radial distortion correction (Figure 5.13(b)) fail to produce a good reprojection for the non-parallel case.

In particular, the noisier reprojections of focal length correction (Figure 5.13(a)) and radial distortion correction (Figure 5.13(b)) stem from two sources of error. First, the depth hypotheses computed are more erroneous than refractive correction, as can be seen from the higher RMS errors (Figure 5.12(b)). Second, the reprojection itself will be erroneous since we need to take into consideration the fact that a depth hypothesis at a pixel will rarely ever fall on the (non-refracted) ray cast through that pixel.

To emphasize this error, a 3D ray is imaged under the influence of a refractive interface that is parallel (Figure 5.14(a)) and non-parallel (Figure 5.14(b)) to the imaging plane. We visualize the reprojections of the image of this ray when ignoring and compensating for refraction. One can see that ignoring refraction results in an erroneous reprojection in both cases, but especially so in the non-parallel case. The level of error is more significant in the non-parallel case because the refractive distortion is far more significant. Figure 5.14 also shows how ignoring refraction can result in misaligned reprojections, which

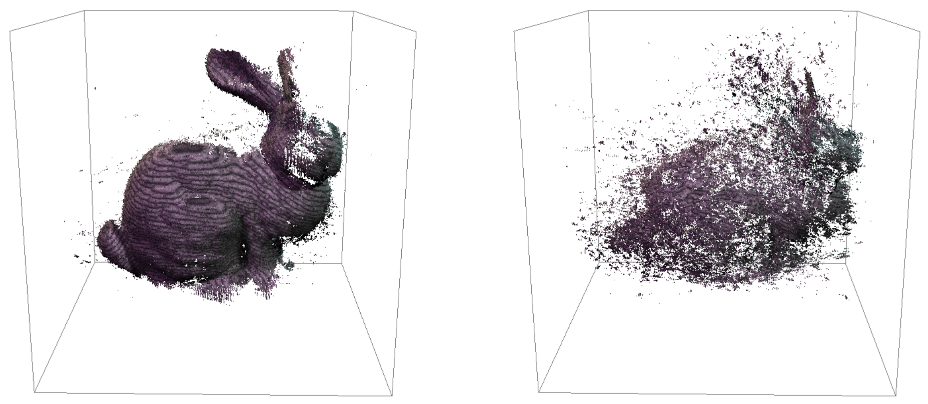


(a) Experiment #1

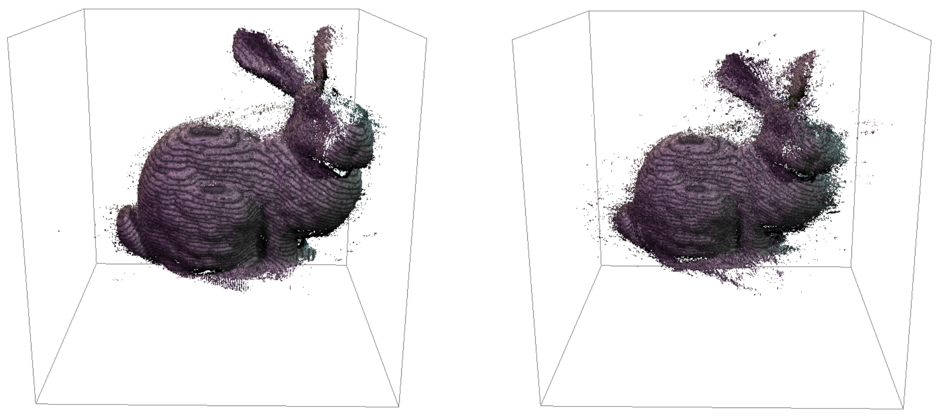


(b) Experiment #2

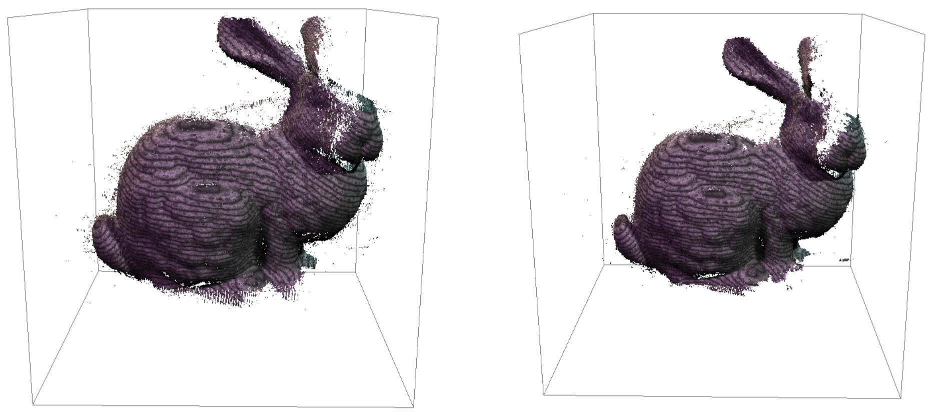
Figure 5.12: Root mean square (RMS) error plots for multi-view experiments on synthetic data.



(a) Focal Length Correction



(b) Radial Distortion Correction



(c) Refractive Correction

Figure 5.13: Reprojection of depth maps from multi-view experiments #1 (left) and #2 (right). Not compensating for refraction when reprojecting depth hypotheses produces misaligned point sets. This misalignment is more pronounced in experiment #2 because the non-parallel interfaces impose larger amounts of refractive distortion.

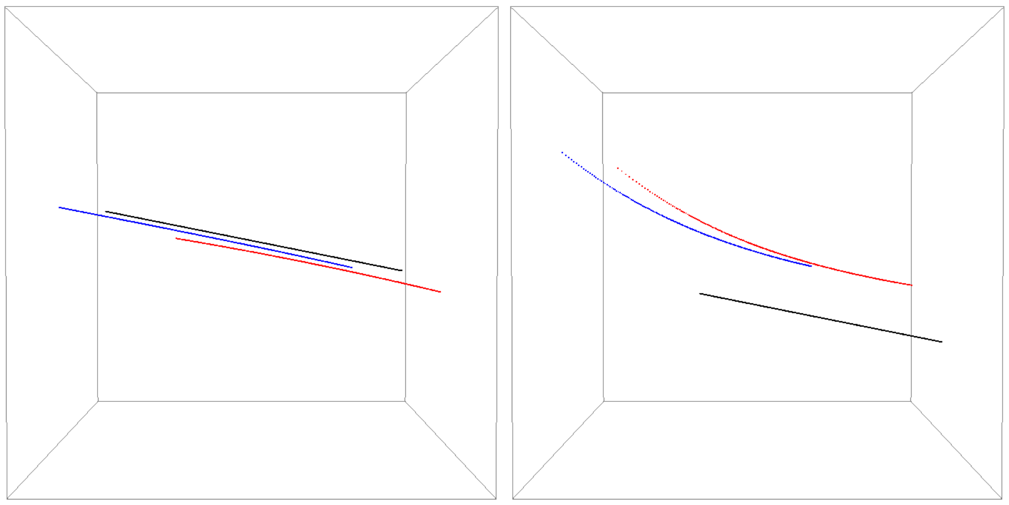

(a) Parallel Interface (b) Non-Parallel Interface

Figure 5.14: Visualizing the reprojected samples of an imaged ray (black) from two views (red/blue) when ignoring a refractive interface. The reprojected samples when ignoring refraction do not line up with the ground truth ray (erroneous reprojection). Also, the reprojected samples do not match each other (misaligned reprojections).

again is more pronounced in the non-parallel case. Misalignment error is especially important in the case of multi-view algorithms that reconstruct the scene with reprojected depth maps (e.g., [6]).

To see how this error changes with respect to a change in the interface's normal, we compute the RMS error of the reprojected samples using (5.1). In particular, β corresponds to the samples taken from the 3D ray and $\hat{\beta}$ the reprojected samples. The squared difference of these two is computed using a squared L_2 norm (i.e., the squared Euclidean distance). First, we note how the minimum occurs at $\sim 7^\circ$ and not at 0° (i.e., parallel interface). This characteristic is specific to the ray that we used to produce this figure, whose image is straightest when the interface is at that angle. In general, the error grows as the angle between the normal of the refractive interface and the optical axis increases.

Finally, our real-world experiments with a multi-view system have been less successful. In particular, we have a setup in which cameras are very close to the refractive interface (Figure 4.2). We have found that calibrating the refractive interfaces under this setup has been a difficult task. Figure 5.16 shows results from one of the data sets captured with this array. The poorly calibrated interfaces resulted in radial distortion correction providing slightly more depth

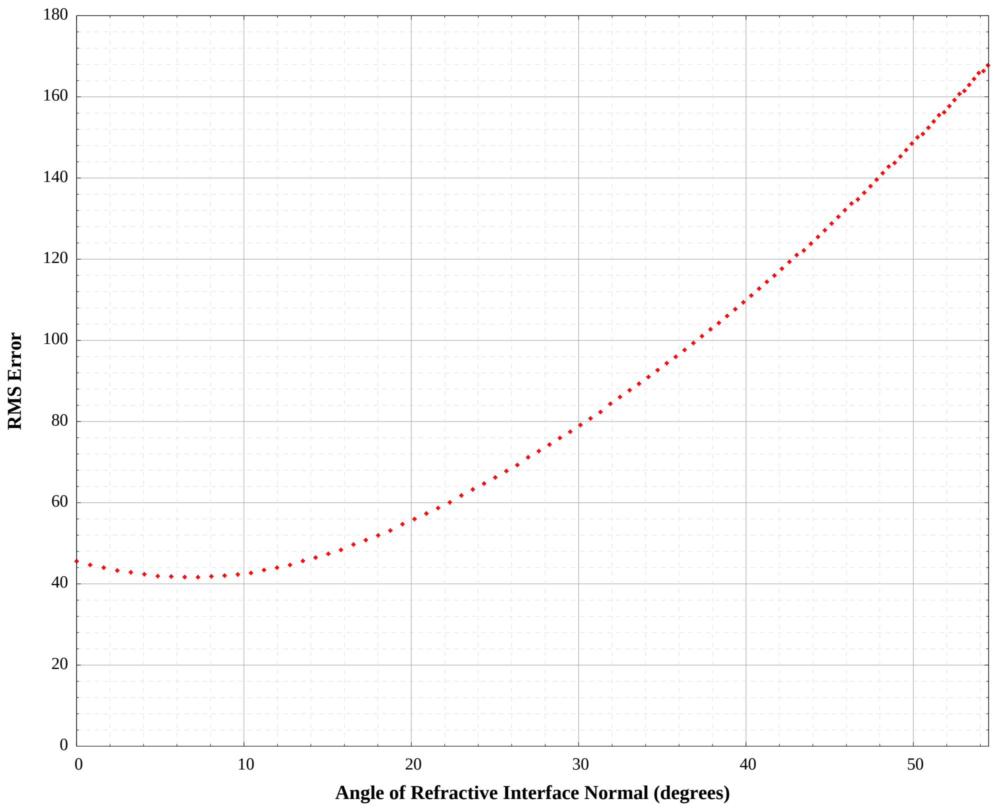
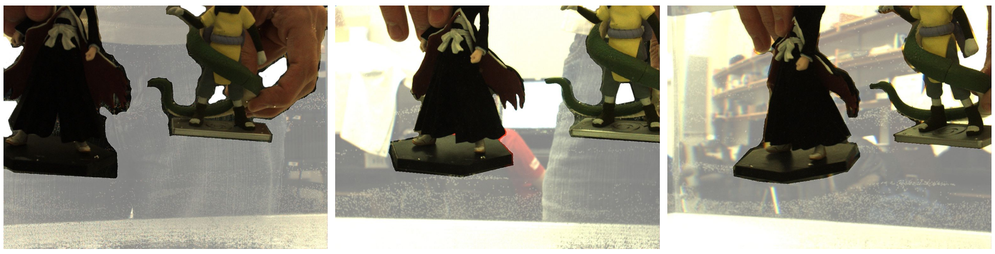
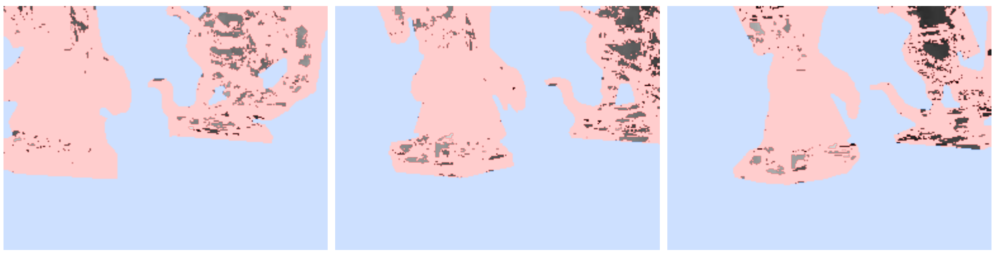


Figure 5.15: RMS error of reprojection when ignoring refraction. The angle of the refractive interface normal is with respect to the optical axis, in the direction of the image diagonal. The leftmost error value corresponds to Figure 5.14(a) and the rightmost to Figure 5.14(b).

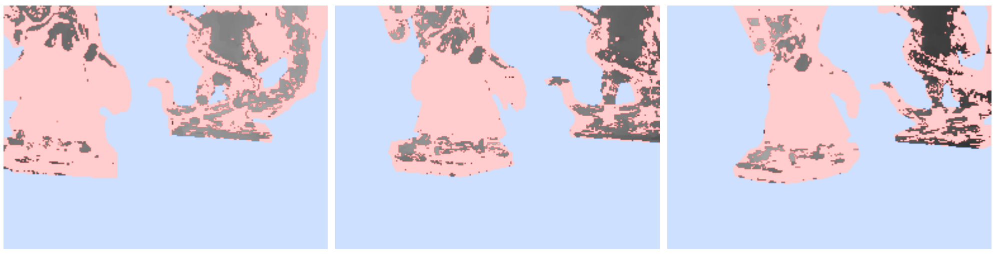
hypotheses than refractive correction, and a better reprojection (Figure 5.17). Given our tank setup, the range of depths that objects can fall within is very small. As mentioned in Section 2.3, a small variation in observed depths is a scenario in which radial distortion can provide satisfactory results.



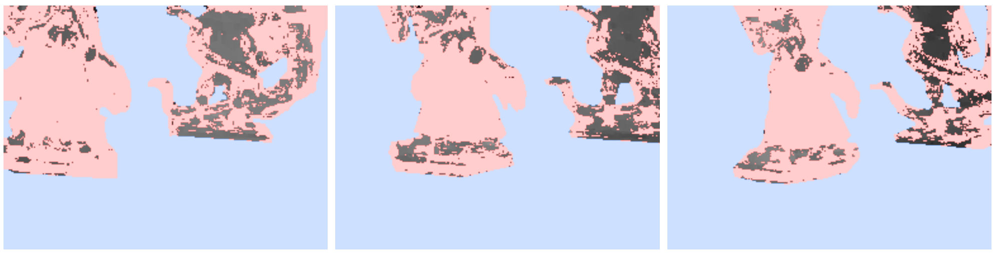
(a) Original Images



(b) Focal Length Correction

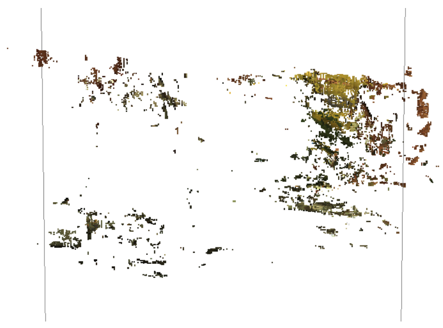


(c) Radial Distortion Correction

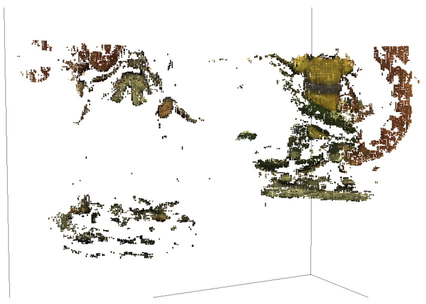


(d) Refractive Correction

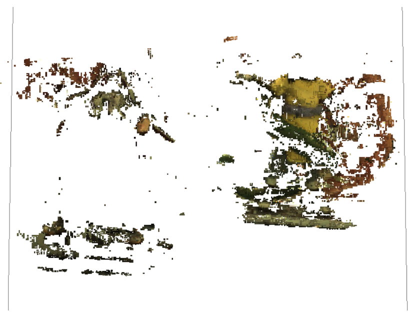
Figure 5.16: Multi-view stereo experiment #3, using real data. Pixel brightness corresponds to depth, with white being the farthest and black being the closest. Masked-out pixels are blue and invalidated pixels are pink. A poorly calibrated set of interfaces prevented refractive correction from obtaining results that were better than radial distortion correction.



(a) Focal Length Correction



(b) Radial Distortion Correction



(c) Refractive Correction

Figure 5.17: Reprojection of depth maps from multi-view experiment #3. Given the poorly calibrated set of interfaces, the reprojection of refractive correction is no better than that of radial distortion correction.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have presented a novel approach to stereo matching in an underwater environment. In particular, this approach consists of two major components:

- a method for calibrating the parameters of a refractive interface, and
- a method for performing stereo matching in which epipolar geometry is represented using a physical model of refraction.

Our calibration method minimizes an approximate reprojection error between point correspondences of two views. From this we obtain parameters of the planar refractive interfaces local to each camera. Given these parameters we can then provide a close approximation of the epipolar geometry imposed by the refractive interfaces. We do so by first casting a ray through a pixel in one image and refracting it against its associated interface. This is followed by projecting points sampled on this ray into the other view. We then use these projected samples to obtain a piecewise linear approximation of the epipolar curve.

Through various experiments we have shown the effectiveness of our system, and how important it is to use a physical model for refraction rather than approximations. Our stereo matching method consistently provides a larger set of depth hypotheses in all experiments.

6.2 Future Work

Although we have provided convincing results for why it is necessary to model refraction properly, there are currently some limitations to our system.

First, calibration is quite sensitive to initial estimates. Oftentimes the calibration would produce spurious results when initialized with values that were relatively far from the actual values. We have found that the error function contains many local minima, making it easy for the optimization to get trapped. Also, we have found parameter vectors which give a smaller total error than parameter vectors that are more physically correct. We believe that further investigation into other optimization strategies is necessary to improve the robustness of the calibration. Other possibilities include:

- Introducing a prior into the optimization to constrain parameter selection.
- Intelligently selecting multiple samples throughout the parameter space, optimizing with each of these samples as initial estimates and taking the best result.

Second, calibration is currently formulated for pairs of cameras. This pairwise process makes it tedious to calibrate refractive interfaces in a multi-view system. Modifying the existing formulation to fit into a bundle adjustment problem would make this process simpler. Also, as mentioned in Section 4.1, it is important to capture calibration data close to the interface. Requiring feature correspondences makes this difficult, if not impossible, when cameras are close to the interface. Another interesting avenue of research would be to investigate single-view calibration, eliminating the need for feature correspondences.

Finally, although it works well, our approach to stereo matching requires sufficiently dense sampling of a ray. A more effective approach would be to have an analytic model of the epipolar curve in image-space, and sample that curve accordingly. Chari and Sturm have provided a closed-form solution to a situation in which a stereo pair have a shared refractive interface [9]. It would be interesting to investigate their work and to see if it can be extended to the more general model (i.e., non-shared refractive interface) proposed in this thesis.

Bibliography

[1] Distance between lines and segments with their closest point of approach. http://softsurfer.com/Archive/algorithm_0106/algorithm_0106.htm.

[2] OpenCV. http://opencv.willowgarage.com/.

[3] POV-Ray. http://www.povray.org/.

[4] Kristian Ambrosch and Wilfried Kubinger. Accurate hardware-based stereo vision. *Computer Vision and Image Understanding*, 114(11):1303–1316, 2010.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1222–1239, 2001.

[6] D. Bradley, T. Boubekeur, and W. Heidrich. Accurate multi-view reconstruction using robust binocular stereo and surface meshing. *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[7] D. F. Campbell, G. Vogiatzis, C. Hernandez, and R. Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. *European Conference on Computer Vision*, 5302:766–779, 2008.

[8] NEPTUNE Canada. NEPTUNE canada. http://www.neptunecanada.ca/, 2011.

[9] Visesh Chari and Peter Sturm. Multi-view geometry of the refractive plane. In *British Machine Vision Conference*, 2009.

[10] D. Claus and A.W. Fitzgibbon. A rational function lens distortion model for general cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 213–219, 2005.

[11] P.F. Felzenszwalb and D.R. Huttenlocher. Efficient belief propagation for early vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 261– 268, 2004.

[12] R. Ferreira, J. P. Costeira, and J.A. Santos. Stereo reconstruction of a submerged scene. In *Iberian Conference on Pattern Recognition*, 2005.

[13] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(1), 2009.

[14] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12:16–22, 2000.

[15] G. Glaeser and H Schröcker. Reflections on refractions. *Journal for Geometry and Graphics*, 4(1):1–18, 2000.

[16] Andrew S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press Ltd., 1989.

[17] N. R. Gracias, S. van der Zwaan, A. Bernardino, and J. Santos-Victor. Mosaic-based navigation for autonomous underwater vehicles. *IEEE Journal of Oceanic Engineering*, 28(4):609–624, 2003.

[18] R. I. Hartley and P. Sturm. Triangulation. In *ARPA Image Understanding Workshop*, pages 957–966, 1994.

[19] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[20] Heiko Hirschmüller and Daniel Scharstein. Evaluation of cost functions for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[21] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann. Local stereo matching using geodesic support weights. In *IEEE International Conference on Image Processing*, pages 2093–2096, 2009.

[22] Ryohei Kawai, Atsushi Yamashita, and Toru Kaneko. Three-dimensional measurement of objects in water by using space encoding method. In *IEEE International Conference on Robotics and Automation*, pages 2830–2835, 2009.

[23] Donna M. Kocak, Fraser R. Dalgleish, Frank M. Caimi, and Yoav Y. Schechner. A focus on recent developments and trends in underwater imaging. *Marine Technology Society Journal*, 42(1):52–67, 2008.

[24] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

[25] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusions using graph cuts. In *IEEE International Conference on Computer Vision*, volume 2, pages 508–515, 2001.

[26] C. Kunz and H. Singh. Hemispherical refraction and camera calibration in underwater vision. In *OCEANS 2008*, pages 1–7, 2008.

[27] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *IEEE International Conference on Computer Vision*, pages 1–8, 2007.

[28] J. Lavest, G. Rives, and J. Lapreste. Underwater camera calibration. *Lecture notes in computer science*, pages 654–668, 2000.

[29] J. P. Lewis. Fast normalized cross-correlation. In *Vision Interface*, pages 120–123. Canadian Image Processing and Pattern Recognition Society, 1995.

[30] Rongxin Li, Haihao Li, Weihong Zou, R. G. Smith, and T. A. Curran. Quantitative photogrammetric analysis of digital underwater video imagery. *IEEE Journal of Oceanic Engineering*, 22(2):364–375, 1997.

[31] N. J. W. Morris and K. N. Kutulakos. Dynamic refraction stereo. In *IEEE International Conference on Computer Vision*, 2005.

[32] S. G. Narasimhan and S. K. Nayar. Structured light methods for underwater imaging: Light stripe scanning and photometric stereo. In *OCEANS*, volume 3, pages 2610–2617, 2005.

[33] Nathalie Pessel, Jan Opderbecke, and Marie-José Aldon. Camera self-calibration in underwater environment. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2003.

[34] Jean-Philippe Pons, Renaud Keriven, and Olivier Faugeras. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. *International Journal of Computer Vision*, 72(2):179–193, 2007.

[35] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.

[36] J. P. Queiroz-Neto, R. Carceroni, W. Barros, and M. Campos. Underwater stereo. In *Brazilian Symposium on Computer Graphics Image Processing*, pages 170–177, 2004.

[37] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2002.

[38] Y. Y. Schechner and N. Karpel. Clear underwater vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 536–543, 2004.

[39] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 519–528, 2006.

[40] M. R. Shortis and E. S. Harvey. Design and calibration of an underwater stereo-video system for the monitoring of marine fauna populations.

International Archives Photogrammetry and Remote Sensing, 32(5):792–799, 1998.

[41] H. Singh, J. Howland, and O. Pizarro. Advances in large-area photomosaicking underwater. *IEEE Journal of Oceanic Engineering*, 29(3):872–886, 2004.

[42] Y. Swirski, Y. Y. Schechner, B. Herzberg, and S. Negahdaripour. Stereo from flickering caustics. In *IEEE International Conference on Computer Vision*, pages 205–212, 2009.

[43] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1068–1080, 2008.

[44] today.mun.ca. $10.1 million in federal funds for research at memorial. http://today.mun.ca/news.php?news_id=6235, 2011.

[45] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, pages 839–846, 1998.

[46] T. Treibitz, Y. Y. Schechner, and H. Singh. Flat refractive geometry. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2008.

[47] A.O. Ulusoy, F. Calakli, and G. Taubin. One-shot scanning using De Bruijn spaced grids. In *IEEE International Conference on Computer Vision Workshops*, pages 1786–1792, 2009.

[48] VENUS. What is VENUS? http://venus.uvic.ca/discover-venus/what-is-venus/, 2011.

[49] G. Vogiatzis, C. Hernandez Esteban, P. H. S. Torr, and R. Cipolla. Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12), 2007.

[50] H. Vu, R. Keriven, P. Labatut, and J. P Pons. Towards high-resolution large-scale multi-view stereo. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[51] M.J. Wainwright, T.S. Jaakkola, and A.S. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005.

[52] Yair Weiss. Belief propagation and revision in networks with loops. Technical report, 1997.

[53] Wikipedia. Cross-correlation. http://en.wikipedia.org/wiki/Cross-correlation.

[54] Yoram Yekutieli, Rea Mitelman, Binyamin Hochner, and Tamar Flash. Analyzing octopus movements using three-dimensional reconstruction. *Journal of Neurophysiology*, 98:1775–1790, 2007.

[55] Kuk-Jin Yoon and In So Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):650–656, 2006.

[56] Li Zhang, B. Curless, and S.M. Seitz. Rapid shape acquisition using color structured light and multi-pass dynamic programming. In *International Symposium on 3D Data Processing Visualization and Transmission*, pages 24–36, 2002.

[57] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Appendix A

C++ Code For Normalized Cross-Correlation

C++ code for computing normalized cross-correlation [29] with support for pixel-based weighting within a window of radius `WINDOW_RADIUS`.

```
double cost_ncc(
    const Image &img1, const Image &img2,
    int x1, int y1, int x2, int y2)
{
  // Find neighboudhood means
  double meanL = 0, meanR = 0;
  double totalWeight = 0.0;
  for(int row = -WINDOW_RADIUS; row <= WINDOW_RADIUS; ++row) {
    for(int col = -WINDOW_RADIUS; col <= WINDOW_RADIUS; ++col) {
      const RGB &lrgb = img1.pixel(x1 + col, y1 + row);
      if(!lrgb.isValid()) continue;

      const RGB &rrgb = img2.pixel(x2 + col, y2 + row);
      if(!rrgb.isValid()) continue;

      const double weight = weight(row, col);
      if(weight > 1e-10) {
        meanL += weight*lrgb.toGray();
        meanR += weight*rrgb.toGray();
        totalWeight += weight;
      }
    }
  }

  // No weights at all, consider this a non match
  if(totalWeight < 1e-10)
    return 0;

  meanL /= totalWeight;
  meanR /= totalWeight;

  // Compute squares and variances
  double sum1 = 0, sum2 = 0, sum3 = 0;
  for(int row = -WINDOW_RADIUS; row <= WINDOW_RADIUS; ++row) {
    for(int col = -WINDOW_RADIUS; col <= WINDOW_RADIUS; ++col) {
      const RGB &lrgb = img1.pixel(x1 + col, y1 + row);
      if (! lrgb . isValid () ) continue;

      const RGB &rrgb = img2.pixel(x2 + col, y2 + row);
      if (! rrgb . isValid () ) continue;

      const double weight = weight(row, col);
      if (weight > 1e−10) {
        const double pixel_gray_l  = weight∗lrgb.toGray();
        const double pixel_gray_r  = weight∗rrgb.toGray();
        sum1 += ( pixel_gray_l  − meanL)∗(pixel_gray_r  − meanR);
        sum2 += ( pixel_gray_l  − meanL)∗(pixel_gray_l  − meanL);
        sum3 += ( pixel_gray_r  − meanR)∗(pixel_gray_r  − meanR);
      }
    }
  }

  // Prevent a divide−by−zero scenario
  if (sum2 ∗ sum3 < 1e−10)
    return 0;

  // Cross−correlation value
  return sum1 / std :: sqrt (sum2 ∗ sum3);
}
```

Appendix B

C++ Code For Refractive Projection

C++ code for projecting a point p onto a refractive interface.

```
using namespace Eigen;

bool refractive_projection (Vector3d &p, const Plane3d &P, double n) {
  // Some values we will need
  const Vector3d proj = project(p, P.normal());
  const double y = (p - proj).y();
  const double z = proj.norm();
  const double r = (p - proj).norm();
  const double d = P.x0().norm();
  const double rr = r*r, nn = n*n, dd = d*d;

  // The direction which when scaled by the proper root will
  // give us the projected point under refraction
  Vector3d dir = p - proj;
  dir.normalize();

  // Find roots of quartic
  double roots[4];
  findRoots(
    nn - 1,
    -2*r*(nn - 1),
    rr*(nn - 1) + dd*nn - (z - d)*(z - d),
    -2*dd*nn*r,
    dd*nn*rr,
    roots[0], roots[1], roots[2], roots[3] );

  // Find the root such that its corresponding projected point
  // has a y-coordinate that is in the range [0, p_y]
  for(int index = 0; index < 4; ++index) {
    if(!std::isnan(roots[index])) {
      const Vector3d pp = roots[index]*dir;
      const double py = pp.y();
      if(py > -1e-3 && y > -1e-3) {
        if(py < y + 1e-3) {
          p = pp + P.x0();
          return true;
        }
      } else if (py < 1e-3 && y < 1e-3) {
        if (y < py + 1e-3) {
          p = pp + P.x0();
          return true;
        }
      }
    }
  }

  return false;
}
```

Appendix C

C++ Code For Triangulating a 3D Point

C++ code for triangulating a point from a set of 2D projections in several views. This is an implementation of the *Iterative-LS* method presented by Hartley and Sturm [18].

```
using namespace Eigen;

const double NaN = std:: numeric_limits <double>::quiet_NaN();

Vector3d  triangulate (const std :: vector<CameraPtr> &cameras,
                const std :: vector<Vector2d> &pts,
                const std :: vector<bool> &mask)
{
  // Find the number of views that have a point
  size_t num_points = 0;
  for( size_t index = 0; index < mask.size() ; ++index)
    if (mask[index])
      ++num_points;

  if (num_points < 2)
    return Vector3d(NaN, NaN, NaN);

  // Iterative −LS method
  MatrixXd A(2∗num_points, 3);
  VectorXd b(2∗num_points);
  Vector4d x(0, 0, 0, 1);
  for( int iteration = 0; iteration < 10; ++ iteration ) {
    for( size_t index = 0, tindex = 0; index < pts. size () ; ++index) if (mask[index])↩
        {
      const Vector2d pt = pts [index ];
      const ProjMat &P = cameras[index]−>P();

      const double weight = ( iteration == 0 ? 1.0 : 1.0 / P.row(2).dot(x));
      const Vector4d t1 = weight∗(pt.x() ∗ P.row(2) − P.row(0)). start <4>();
      const Vector4d t2 = weight∗(pt.y() ∗ P.row(2) − P.row(1)). start <4>();

      A.row(tindex + 0) = t1. start (3);
      A.row(tindex + 1) = t2. start (3);
      b[tindex + 0] = -t1[3];
      b[tindex + 1] = -t2[3];

      tindex += 2;
    }

    // Solve the linear system
    Eigen::Vector3d xt;
    A.svd().solve(b, &xt);

    // Break if new solution isn't much different from previous
    if( iteration > 0 && (x.start<3>() - xt).squaredNorm() < 1e-10)
      break;

    x.start<3>() = xt;
    break;
  }

  return x.start<3>();
}
```

Appendix D

Computing The Midpoint of Two Rays

The midpoint of two rays can intuitively be regarded as the point which is as close as possible to both rays. More specifically, consider rays $R_1 = \{s_1, \vec{d_1}\}, R_2 = \{s_2, \vec{d_2}\}$, where s_i are the sources of the rays and $\vec{d_i}$ are their directions. We compute the following values [1]:

$$\begin{aligned} w_0 &= s_1 - s_2 && \text{(D.1)} \\ a &= \vec{d_1} \cdot \vec{d_1} && \text{(D.2)} \\ b &= \vec{d_1} \cdot \vec{d_2} && \text{(D.3)} \\ c &= \vec{d_2} \cdot \vec{d_2} && \text{(D.4)} \\ d &= \vec{d_1} \cdot w_0 && \text{(D.5)} \\ e &= \vec{d_2} \cdot w_0 && \text{(D.6)} \\ t_1 &= \frac{be - cd}{ac - b^2} && \text{(D.7)} \\ t_2 &= \frac{ae - bd}{ac - b^2}. && \text{(D.8)} \end{aligned}$$

If we consider the extensions of the rays R_1, R_2 to infinite lines, the point p_1 on R_1 which is as close as possible to R_2 is computed from t_1:

$$p_1 = s_1 + t_1\vec{d_1}. \quad \text{(D.9)}$$

Similarly, the point p_2 on R_2 which is as close as possible to R_1:

$$p_2 = s_2 + t_2\vec{d_2}. \quad \text{(D.10)}$$

The midpoint m of R_1 and R_2 is then the midpoint of points p_1 and p_2:

$$m = \frac{p_1 + p_2}{2}. \quad \text{(D.11)}$$

Note how we consider the extension of the rays to infinite lines. In the case that $t_1 < 0$ or $t_2 < 0$ we are outside of the domain of the ray. Special care needs to be taken for these boundary cases [1].

C++ code corresponding to the above:

```
Point midpoint(const Ray3d &R1, const Ray3d &R2) const {
  Point w0 = R1.source() - R2.source();

  double a = R1.direction().dot(R1.direction());
  double b = R1.direction().dot(R2.direction());
  double c = R2.direction().dot(R2.direction());
  double d = R1.direction().dot(w0);
  double e = R2.direction().dot(w0);

  double den = 1.0 / (a*c - b*b);
  double t1 = (b*e - c*d) * den;
  double t2 = (a*e - b*d) * den;

  Point p1 = R1.source() + t1*R1.direction();
  Point p2 = R2.source() + t2*R2.direction();
  return (p1 + p2)*0.5;
}
```

Appendix E

C++ Code For Computing Geodesic Support Weights

C++ code for computing geodesic support weights [21] in a window with a radius of `WINDOW_RADIUS`.

```
namespace {
  // Sigma value used in weighting
  const double GEODESIC_SIGMA = 25.0;

  // Num iterations for sweeping shortest-path method
  const int NUM_ITERS = 3;

  // Offsets kernel for sweeping shortest-path method
  const int KERNEL_SIZE = 8;
  const double K1[] = {-1, -1, 0, -1, 1, -1, -1, 0}; // forward pass
  const double K2[] = {-1,  1, 0,  1, 1,  1,  1, 0}; // backward pass
}

// 2D array type containing weights
typedef std::vector<std::vector<double> > Weights;

Weights compute_geodesic_weights(const VectorImage &img, int cx, int cy) {
  const int WINDOW_SIZE = 2*WINDOW_RADIUS + 1;

  // Initialize distances to large values, and center pixel to zero
  Weights weights(WINDOW_SIZE);
  for(int y = 0; y < WINDOW_SIZE; ++y)
    std::fill(weights[y].begin(), weights[y].end(), 1000000.0);

  weights[WINDOW_RADIUS][WINDOW_RADIUS] = 0.0;

  // Approximate geodesic distance using sweeping method
  for(int iter = 0; iter < NUM_ITERS; ++iter) {
    // Forward pass
    for(int y = -radius; y <= WINDOW_RADIUS; ++y) {
      for(int x = -WINDOW_RADIUS; x <= WINDOW_RADIUS; ++x) {
        const RGBA &rgb1 = img.pixel(cx + x, cy + y);
        if(!rgb1.isValid())
          continue;
        double &weight = weights[y + WINDOW_RADIUS][x + WINDOW_RADIUS];
        for(int ind = 0; ind < KERNEL_SIZE; ind += 2) {
          int dx = K1[ind + 0];
          int dy = K1[ind + 1];
          if(x + dx > WINDOW_RADIUS || y + dy > WINDOW_RADIUS || x + dx < ↩
              −WINDOW_RADIUS || y + dy < −WINDOW_RADIUS)
            continue;

          RGBA rgb2 = img.pixel(cx + x + dx, cy + y + dy);
          if(rgb2.isValid()) {
            rgb2 −= rgb1;
            double diff = std::sqrt(rgb2.r∗rgb2.r + rgb2.g∗rgb2.g + rgb2.b∗rgb2.b)↩
                ;
            double cost = weights[y + dy + WINDOW_RADIUS][x + dx + ↩
                WINDOW_RADIUS];
            weight = std::min(weight, cost + diff);
          }
        }
      }
    }

    // Backward pass
    for(int y = WINDOW_RADIUS; y >= −WINDOW_RADIUS; −−y) {
      for(int x = WINDOW_RADIUS; x >= −WINDOW_RADIUS; −−x) {
        const RGBA &rgb1 = img.pixel(cx + x, cy + y);
        if(!rgb1.isValid())
          continue;

        double &weight = weights[y + WINDOW_RADIUS][x + WINDOW_RADIUS];
        for(int ind = 0; ind < KERNEL_SIZE; ind += 2) {
          int dx = K2[ind + 0];
          int dy = K2[ind + 1];
          if(x + dx > WINDOW_RADIUS || y + dy > WINDOW_RADIUS || x + dx < ↩
              −WINDOW_RADIUS || y + dy < −WINDOW_RADIUS)
            continue;

          RGBA rgb2 = img.pixel(cx + x + dx, cy + y + dy);
          if(rgb2.isValid()) {
            rgb2 −= rgb1;
            double diff = std::sqrt(rgb2.r∗rgb2.r + rgb2.g∗rgb2.g + rgb2.b∗rgb2.b)↩
                ;
            double cost = weights[y + dy + WINDOW_RADIUS][x + dx + ↩
                WINDOW_RADIUS];
            weight = std::min(weight, cost + diff);
          }
        }
      }
    }
  }

  // Exponential weighting
  for(int y = 0; y < WINDOW_SIZE; ++y)
    for(int x = 0; x < WINDOW_SIZE; ++x)
      weights[y][x] = std::exp(−weights[y][x] / GEODESIC_SIGMA);

  return weights;
}
```