Real-time stereo reconstruction using hierarchical dynamic programming and LULU filtering

by

François Singels

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science

at

Stellenbosch University

Department of Mathematical Sciences

Faculty of Science

Supervisor: Dr Willie Brink
Co-supervisor: Prof. Ben Herbst

Date: March 2010

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 19 February 2010

Abstract

In this thesis we consider the essential topics relating to stereo-vision and the correspondence problem in general. The aim is to reconstruct a dense 3D scene from images captured by two spatially related cameras. Our main focus, however, is on speed and real-time implementation on a standard desktop PC. We wish to use the CPU to solve the correspondence problem and to reserve the GPU for model rendering. We discuss three fundamental types of algorithms and evaluate their suitability to this end. We eventually choose to implement a hierarchical version of the dynamic programming algorithm, because of the good balance between accuracy and speed. As we build our system from the ground up we gradually introduce necessary concepts and established geometric principles, common to most stereo-vision systems, and discuss them as they become relevant. It becomes clear that the greatest weakness of the hierarchical dynamic programming algorithm is scanline inconsistency. We find that the one-dimensional LULU-filter is computationally inexpensive and effective at removing outliers when applied across the scanlines. We take advantage of the hierarchical structure of our algorithm and sub-pixel refinement to produce results at video rates (roughly 20 frames per second). A $3D$ model is also constructed at video rates in an on-line system with only a small delay between obtaining the input images and rendering the model. Not only is the quality of our results highly competitive with those of other state of the art algorithms, but the achievable speed is also considerably faster.

Opsomming

In hierdie tesis beskou ons die noodsaaklike onderwerpe wat in die algemeen verband hou met stereovisie en die ooreenstemmingsprobleem. Die mikpunt is om 'n digte 3D toneel te rekonstrueer vanaf beelde wat deur twee ruimtelik-verwante kameras vasgelê is. Ons hoofdoel is egter spoed, en intydse implementering op 'n standaard rekenaar. Ons wil die SVE (CPU) gebruik om die ooreenstemmingsprobleem op te los, en reserveer die GVE (GPU) vir model-beraping. Ons bespreek drie fundamentele tipes algoritmes en evalueer hul geskiktheid vir hierdie doel. Ons kies uiteindelik om 'n hiërargiese weergawe van die dinamiese programmeringsalgoritme te implementeer, as gevolg van die goeie balans tussen akkuraatheid en spoed. Soos wat ons ons stelsel van die grond af opbou, stel ons geleidelik nodige konsepte voor en vestig meetkundige beginsels, algemeen tot meeste stereovisie stelsels, en bespreek dit soos dit toepaslik word. Dit word duidelik dat skandeerlyn-strydigheid die grootste swakheid van die hiërargiese dinamiese programmeringsalgoritme is. Ons vind dat die een-dimensionele LULU-filter goedkoop is in terme van berekeninge, en effektief aangewend kan word om uitskieters te verwyder as dit dwarsoor skandeerlyne toegepas word. Ons buit die hiërargiese struktuur van ons algoritme uit en kombineer dit met sub-piksel verfyning om resultate te produseer teen video tempo (ongeveer 20 raampies per sekonde). 'n $3D$ model word ook gekonstrueer teen video tempo in 'n stelsel wat aanlyn loop, met slegs 'n klein vertraging tussen die verkryging van die intree-beelde en die beraping van die model. Die kwaliteit van ons resultate is nie net hoogs mededingend met dié van die heel beste algoritmes nie, maar die verkrygbare spoed is ook beduidend vinniger.

Acknowledgments

I would like to thank the following people, organizations and entities:

- God, for giving me the gift of sight and the opportunity to study it in an extraordinary way.
- My family and girlfriend, Esterbeth (Zep) le Roux, for their love and support and above all their patience.
- Prof. Ben Herbst, my co-supervisor, for his encouragement and words of advice.
- The National Research Foundation, for their financial support.
- And a special thanks to my supervisor, Willie Brink, for being a friend as well as a mentor.

Contents

List of Figures

List of Tables

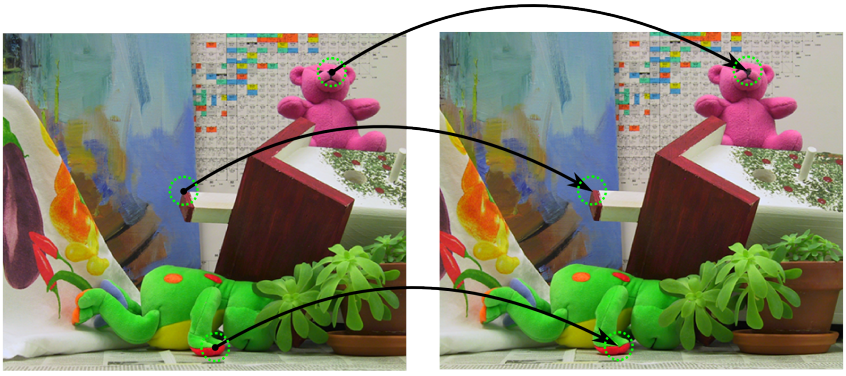
Chapter 1

Introduction

Computer-vision is a broad field of study, relating to any situation where a computer is called upon to interpret an image, or set of images, and make some conclusions base on what it sees. In this thesis we are interested in the stereo-vision facet of computer-vision. In stereo-vision (also called binocular-vision) the computer is given two pictures of a scene, each taken from a slightly different angle, and asked to reconstruct a $3D$ model. This is done by trying to find a match for every pixel in the reference image in the corresponding image, as **Figure** 1.1 illustrates, then using these matches to triangulate their coordinates and construct a $3D$ model, illustrated in **Figure** 1.2. This should be easy since we, as humans, do it all the time. How hard can it be?

Unfortunately it turned out to be a much more complex problem than many people thought, especially once we realise the enormous amount of brain power we humans require to process what we see (roughly a quarter of the neocortex [1]). Humans also use heuristic knowledge and their other senses to more clearly perceive the world around them. After all, we had millions of years to get our eyes the way they are today. Computers, on the other hand, are not even a century old yet. Nevertheless, the dream of making computers see is alive and pursued with great zeal.

The quest so far has been far from fruitless and many systems have been developed for stereo-vision. But, as with most resource intensive computer programs,



(a) (b)

Figure 1.1: Two images used in stereo-vision: (a) left image, (b) right image. The idea is to find a matching pixel in (b) for every pixel in (a). (Example taken from the Middlebury dataset.)

Figure 1.2: $3D$ reconstruction of the scene in **Figure** 1.1.

there is sharp trade-off between accuracy and speed. Ideally we would like to match, or even surpass, the limits of human vision [2].

1.1 Applications

Stereo-vision enables us to navigate our surroundings, avoid obstacles and recognise shapes, without interacting with them directly. Giving machines these abilities will allow them to perform much more complicated tasks with applications in various fields.

Mobile robotics has perhaps the most intuitive applications for stereo-vision, enabling robots to navigate themselves and avoid obstacles autonomously. Simul-

taneous Localisation and Mapping (SLAM) [3, 4] is an extension of this idea where robots orientate themselves in, and create a map of, their environment using cameras. This type of application can also extend to the field of $3D$ surveying [5]. One of the great advantages of vision over a system such as echo-location (or radar) is that it is completely passive, requiring less energy and allowing one to observe without giving away one's position.

Human-computer-interaction [6, 7], where a computer needs to recognise a certain gesture or pose, can benefit from the extra $3D$ information provided by stereo-vision. Sign language can also be more accurately interpreted from a $3D$ model than just a $2D$ image.

Augmented reality [8] can use stereo-vision to insert virtual objects in real world scenes. It can also be used to enhance our own vision, for example to highlight objects of interest in a head mounted display and provide some useful information such as the distance to objects.

1.2 Objectives

The main aim of this study is to build a complete, on-line stereo system that can produce output at video rates and allow for it to be processed further by other applications. We aim to implement this system on a standard desktop PC, to keep the cost low and allow for some extensibility. This will require the process to be very efficient and to not use an excessive amount of resources.

The goal is to use the output in real-time applications which means that speed is of the essence. For humans, a video displayed at 25 to 30 frames per second is fast enough to appear seamless. Real-time also implies that the latency between receiving images and producing the output should be as small as possible.

Although the main emphasis is on speed we do not want to sacrifice too much accuracy in order to obtain it. The advantages of speed will be for naught if the resulting $3D$ model is unintelligible.

1.3 Overview of this study

To reach our goals, we need to "stand on the shoulders of giants" and study what has been done in the past. In **Chapter** 2 we review some of the achievements in the field of stereo-vision and the current state of the art. To decide what approach we will take, several algorithms are considered and a decision is made based on the speed and quality of each.

In **Chapter** 3 we discuss the geometric properties of projective cameras and how the union of two cameras is used to estimate depth. The process of reducing the search area for matching points through epipolar geometry is also covered, as well as camera calibration and triangulation.

The chosen algorithm, hierarchical dynamic programming, and some important concepts relating to stereo algorithms in general are explained in detail, through **Chapter** 4. Multi-level LULU filtering, used to remove outliers and suppress the effects of scanline inconsistency, and the sub-pixel refinement stage are also covered in this chapter.

Chapter 5 showcases our results and evaluates the algorithm's effectiveness using the Middlebury dataset. Finally we draw some conclusions in **Chapter** 6 and present some ideas for future work.

Chapter 2

Literature Review

As mentioned in the introduction, quite a number of stereo algorithms have been proposed already, making the task of sifting through a copious amount of literature difficult. The work presented by D. Scharstein and R. Szeliski in [9], with the accompanying website, `vision.middlebury.edu/stereo`, is a good place to start. Their attempt to bring a measure of order to the multitude of algorithms, each measuring performance in a different manner, has been greatly successful. They proposed a standard method of evaluating the performance of an algorithm so it can be compared to any other algorithm in a sensible and consistent way. Researchers can submit their results to the website and receive a ranking, letting them know how well their algorithm performs relative to others. The sheer number of submissions on the site is a testimony to the importance of their work.

One criticism would be the lack of comparison in speed, since speed is of great importance to many applications. However, this type of comparison would require all algorithms to be implemented on exactly the same machine, which is an almost impossible task. Fortunately, in [9] they do implement some of the more fundamental types of algorithms and report their speeds, giving some idea of what the speed of an algorithm will be, depending on what principles it is based on. For a clearer indication of speed, literature relating to the algorithm in question will have to be consulted, but then one has to take into account the type of machine it was implemented on.

(a) (b) (c) (d)

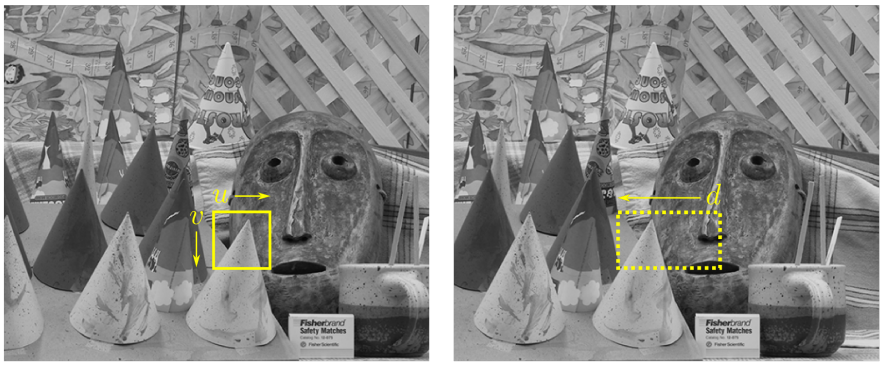
Figure 2.1: Some pictures from the Middlebury dataset with their corresponding ground-truths. Top row: left images. Bottom row: ground truth disparity images. The scenes are referred to as (a) Tsukuba, (b) Venus, (c) Cones and (d) Teddy.

As will be explained in detail in **Chapter** 3, the purpose of stereo-vision algorithms is to obtain a disparity-image, or depth-image. This image contains the disparity, or offset, values of each pixel in the reference image to its matching pixel in the corresponding image. Some examples of these images are given in **Figure** 2.1, where the disparity images in the bottom row are the ground-truth disparity images corresponding to the images in the top row. These ground-truth images are constructed using structured light through the method described in [10] and are used to measure the accuracy of stereo-algorithms.

2.1 Some fundamental methods

According to [9] and [11] stereo methods can be divided roughly into two groups: area based and global optimization methods. Area based methods, see [12, 13, 14, 15], try to find matches by comparing the similarity of small areas around each pixel in both images. Global optimization methods, see [16, 17, 18, 19], take the whole image and try to find optimal matches for all pixels simultaneously, often in an iterative manner.

A detailed discussion and comparison of many algorithms of both groups can be found in [9] and [11]. For a general introduction to the main types of algorithms available we only discuss three methods: sum of the squared difference (SSD),



(a) (b)

Figure 2.2: The SSD approach to stereo matching. (a) A small window of size $u \times v$ around a pixel in the left image is taken. (b) In the right image a matching block is searched for in a horizontal line to find the disparity offset d.

dynamic programming (DP) and graph cut optimization (GC). Many of the state of the art methods, including the one we eventually choose, build upon the same principles as those mentioned above.

2.1.1 Sum of the squared difference

SSD is an area based, or block matching, approach. It takes a small window of size $u \times v$ around a pixel and searches in the corresponding image, along a horizontal line, for a window with similar intensities. At each step the difference Δ of the two windows is calculated using the equation

$$\Delta = \sum_{u,v} (I_L(u,v) - I_R(u-d,v))^2, \tag{2.1}$$

where I_L and I_R represent the intensity values of the left and right images respectively and d is the current disparity offset. The pixel in the right image corresponding to the disparity value with the lowest Δ is then nominated as the best match for the current pixel in the left image. Variations in the difference equation give rise to alternative methods such as: Normalized SSD, Sum of the Absolute Difference (SAD) and Normalized Cross-Correlation (NCC). For some more elaborate measures, including Rank and Census, see [20].

In this type of approach finding the correct match can be tricky, especially in weakly textured areas where there is very little information to distinguish one

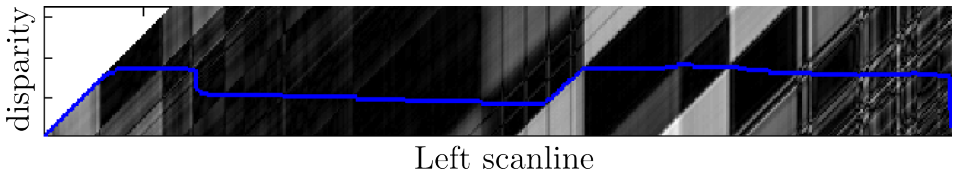


Figure 2.3: An example of a DSI obtained by comparing two scanlines over a range of disparities. The blue line marks the minimum cost path and in turn the disparities corresponding to the left scanline.

pixel from the next. However, it can be implemented quite efficiently. In the naive implementation, with a window size of n and an image size N in pixels, the computational complexity will be $O(NDn)$, where D is the range of considered disparities. This is not very efficient, but by keeping running sums of the block areas redundant calculations can be avoided and the complexity becomes $O(ND)$. Since matches can be computed independently the process can be sped up considerably by making full use of parallel computing on the GPU, see [21]. We will be reserving the GPU for model rendering and future applications, so this step is not considered.

2.1.2 Dynamic programming

Stereo matching through DP, discussed in detail in **Chapter** 4, can be considered a semi-global method. It tries to find the best set of matches for a pair of scanlines (image rows), not just for individual pixels, but not for the whole image either. The process translates to finding the minimum cost path, or path of least resistance, through the disparity space image (DSI) such as the one shown in **Figure** 2.3. This image is constructed by computing the differences of all the pixels in the reference scanline and all the pixels of the corresponding scanline over a range of disparity levels. The differences can be measured in much the same way as in area based methods.

The name dynamic programming comes from a mathematical process through which a problem is solved by breaking it into smaller pieces. The smaller problems are solved first and the results are used to solve the bigger problem. This approach is applied to the problem of finding the minimum cost path. First one defines the way the path can move through the DSI using some constraints. The minimum cost

to get to a certain point is then calculated and used to calculate the minimum cost of getting to the next point of any path that moves through the first point.

One of the great advantages of global methods is that it has an inherent degree of insensitivity to weakly textured areas. In DP the path will not stray far from the disparities at the edges of the textureless area, because that would increase the cost of the path. However, global methods are also inherently more complex. The complexity of the DP algorithm is $O(ND^2)$ which means it is even more sensitive to the disparity search range than SSD.

Considering each pair of scanlines independently is the DP algorithm's greatest strength and weakness. Along scanlines it benefits from a global nature. Across scanlines, however, it suffers from inconsistencies because it does not consider any $2D$ spatial coherence constraints.

2.1.3 Graph cuts

Another way of thinking about the stereo matching problem is by trying to find the maximum flow through a $3D$ graph. The directed graph is defined as $G = (V, E)$, where V is the set of vertices and E is the set of edges. The set of vertices is based on the set of all possible matches. The construction of this set resembles that of the DSI, but in $3D$. This $3D$ structure is what makes GC a truly global method. The set of vertices is defined as

$$V = V^* \cup \{s, t\} \tag{2.2}$$

where s is the source and t is the sink. V^* is defined as

$$V^* = \{(x, y, d), x \in [0, x_{\max}], y \in [0, y_{\max}], d \in [0, d_{\max}]\}, \tag{2.3}$$

where $x_{\max}$ and $y_{\max}$ correspond to the width and height of the images and $d_{\max}$ to the range of disparities. The set of edges is defined as

$$E = \left\{ \begin{array}{ll} (u, v) \in V^* \times V^* & : |u - v| = 1 \\ (s, (x, y, 0)) & : x \in [0, x_{\max}] \\ ((x, y, d_{\max}), t) & : y \in [0, y_{\max}] \end{array} \right\}. \tag{2.4}$$

Figure 2.4 represents the outline of such a graph. Away from s and t it is six-connected and each vertex has a cost associated with it, calculated in much the

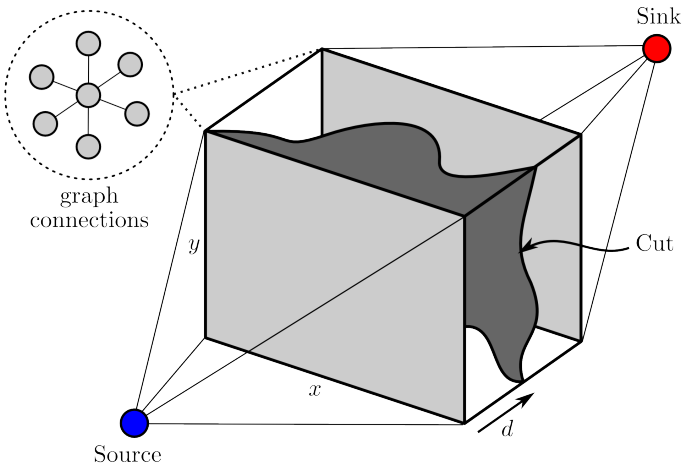


Figure 2.4: Outline of a graph used by the graph cut method.

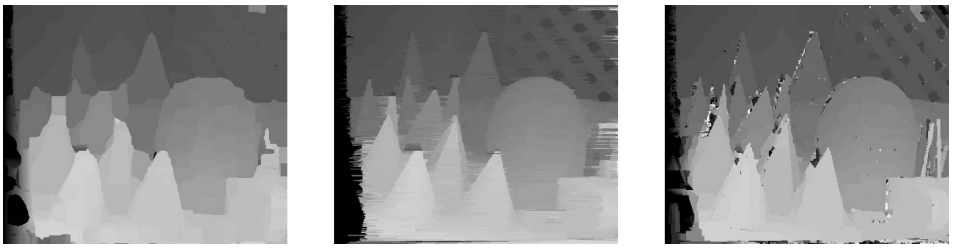
same way as in DP. Each edge has a flow capacity calculated as a function of the costs of the vertices it connects. These capacities limit the flow from the source to the sink.

A cut separates the set of vertices V into two parts, one containing the source and one containing the sink. We now want to find the cut with the lowest capacity, where the capacity of a cut is simply the sum of the edge capacities that define the cut. This minimum cut corresponds to maximizing the flow through the graph, but in stereo-vision it is more intuitive to focus on the minimum cut. The path the cut forms in a single scanline is similar to the path formed through the DSI in DP, but in this case it is extended to $3D$ avoiding the problem of scanline inconsistency.

Algorithms designed for this approach are naturally more resource intensive, requiring large amounts of memory and processing time. The worst case complexity of such an implementation is $O(N^2D^2\log(ND))$, but the algorithm implemented by Roy and Cox in [22] showed an average observed complexity of $O(N^{1.2}D^{1.3})$.

2.2 Comparison

Examples of the results obtained on the Cones scene from **Figure** 2.1 by the three discussed methods are shown in **Figure** 2.5. The SSD struggles on and around object boundaries, because of the area based comparison. Methods of detecting and improving object boundaries have been developed, see [23] and [24], but such



(a) (b) (c)

Figure 2.5: Comparison of results obtained by three different methods on the Cones scene. (a) SSD, (b) DP and (c) GC.

	Tsukuba	Venus	Teddy	Cones	**Average**
GC	5.15 %	4.66 %	22.13 %	13.73 %	**11.4 %**
DP	7.05 %	14.03 %	18.73 %	16.90 %	**14.2 %**
SSD[1]	12.13 %	6.93 %	24.73 %	18.90 %	**15.7 %**

Table 2.1: The percentage bad pixels obtained by each method on four data sets are tabulated to compare their effectiveness. A pixel is considered bad if the measured disparity differs from the ground-truth by more than one.

	Tsukuba	Venus
GC	23.6 s	51.3 s
DP	1.0 s	1.9 s
SSD[1]	1.1 s	1.7 s

Table 2.2: The recorded execution times on two data sets in [9]. Tsukuba has dimensions 384×288 tested over a disparity range of 16 and Venus has dimensions 434×383 tested over a disparity range of 20.

methods require extra computation. The DP results show the typical 'streaky' effect caused by scanline inconsistency, but overall seems to perform quite well. GC produces quality results everywhere, except in occluded regions where some gross errors occur. It is worth noticing that GC is the only one of the three that obtains proper disparities for the sticks in the mug on the right.

A condensed version of the evaluation table published on the Middlebury website mentioned earlier is given in **Table** 2.1. As expected GC performs the best on average, followed by DP and SSD. The lack of a significant difference between DP and SSD suggests how severe the effect of scanline inconsistency can be.

Execution times of the three algorithms were recorded in [9] and shown here in

[1] They implemented a version of the SSD with shiftable windows, producing better results.

Table 2.2. It is clear that, even though computers have advanced much since these recordings where made, GC is not a viable candidate for real-time implementation. We would like to compromise between accuracy and speed by choosing DP, but there is one more problem that needs to be addressed.

When looking at the complexities of the different approaches ($O(DN)$ for SSD, $O(ND^2)$ for DP and $O(N^2D^2\log(ND))$ for GC in the worst case) we see they all depend heavily on the range of disparities. This is a problem, because real-world scenes may consist of objects over a wide range of distances, making it difficult to determine a set range beforehand. For large disparity ranges the complexity of DP becomes less desirable but, in fact, so does that of SSD.

A solution comes in the form of a hierarchical implementation of the DP algorithm (HDP) proposed by Van Meerbergen et al. in [26]. Their version depends on the calculation of disparities at lower sampling levels, to reduce the cost of calculating disparities at higher levels without affecting the quality of the results. They have proven that their algorithm, at the optimum level of down-sampling, reaches an order complexity of

$$O\left(N\left(65-\frac{4071}{(D+5)^2}\right)\right). \tag{2.5}$$

So if D becomes large enough they claim that the algorithm becomes insensitive to the disparity search range. The log complexities of the different algorithms are shown in **Figure** 2.6. From this it is clear that HDP is not only insensitive to disparity search range, but has a complexity rivalling that of SSD.

Based on the low order complexity, insensitivity to disparity search range and the quality of the DP method, HDP is a great candidate for real-time stereo-vision and the one we choose to implement. To make it even more appealing we have to remove the effect of scanline inconsistency. An excellent and efficient method for removing outliers in a one dimensional signal, called the LULU-filter, is discussed in [27] and [28]. We cover the LULU-filter in more detail in **Section** 4.6.

From the next chapter onward we start building our complete stereo system, introducing and explaining concepts as it becomes necessary to do so. A key theme to remember along the way is efficiency, as we try to keep every step as simple and as fast as possible, without sacrificing too much accuracy.

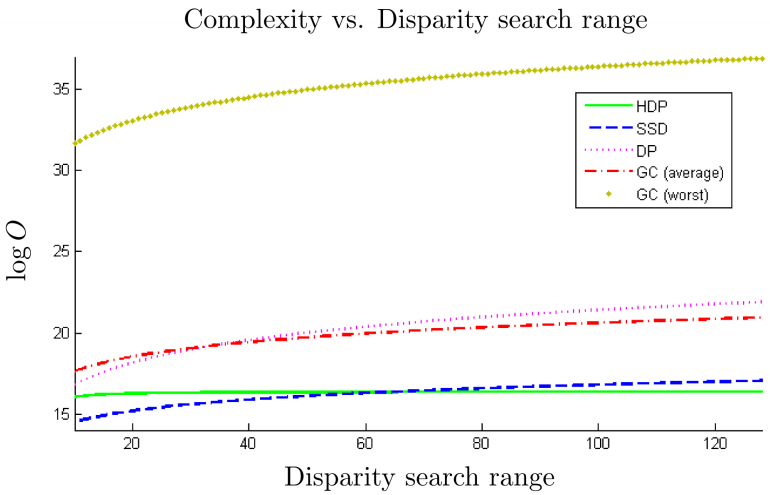


Figure 2.6: The log complexities of all the discussed algorithms versus disparity search range. In this graph $N = 512 \times 384$ and D corresponds to the horizontal axis. It is clear that HDP flattens out while the other algorithms keep on growing. Keep in mind that the actual performance of these algorithms will be effected by a constant offset due to computational overhead.

Chapter 3

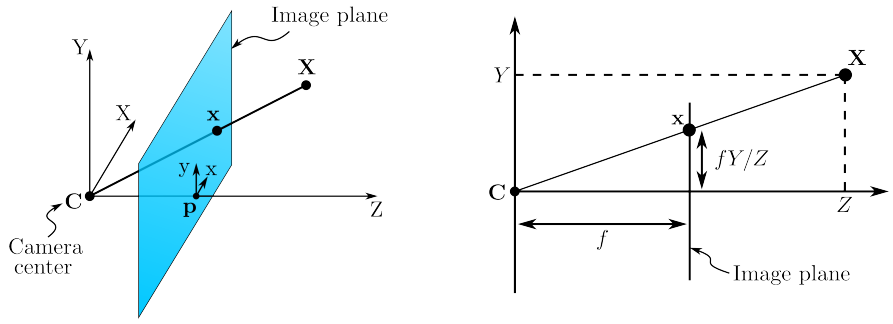
Stereo Geometry

Stereo geometry refers to the geometric relationship between two cameras. In this chapter we discuss the parallel stereographic setup of such a relationship. We start with the geometry of one camera and how to represent it mathematically. Then we take a look at how to calculate these representations for real-world cameras. Next we introduce another camera to the system and discuss the inter-camera relationship via epipolar geometry. Finally we cover image rectification and how this process simplifies the epipolar geometry.

Throughout this chapter we introduce a number of terms that will be used frequently in the rest of the text. These terms are commonly used in stereo-vision literature and will be defined as they become relevant.

3.1 Single camera geometry

The most common model for a CCD camera is the pinhole model represented in **Figure** 3.1(a). This model assumes that every ray of light that enters the camera passes through a single point, the camera centre $\mathbf{C}$. The image is produced as each ray passes through a plane a distance away from the camera centre. The distance from the camera centre to the image plane is called the focal length f. In other words any point $\mathbf{X}$ in $\mathbb{R}^3$ (camera coordinates) projects onto the image plane as $\mathbf{x}$ in $\mathbb{R}^2$ (image coordinates) where the ray from $\mathbf{X}$ through the camera centre $\mathbf{C}$



(a) (b)

Figure 3.1: (a) Projection of a $3D$ coordinate $\mathbf{X}$ onto the image plane using a pinhole camera model. (b) Simple projection using camera coordinates in the Z-Y plane, where f is the focal length.

intersects the image plane.

To represent this projection mathematically, we define $\mathbf{X}$ and $\mathbf{x}$ in homogeneous coordinates and introduce the 3×4 camera matrix P such that

$$\mathbf{x} = \mathrm{P}\mathbf{X}. \tag{3.1}$$

To understand this we start by looking at a very simple projection in the camera's coordinate system. In **Figure** 3.1(b) $\mathbf{C}$ is at the origin and we are looking at the Z-Y plane. By similar triangles the point $(X, Y, Z)^T$ projects to $(fX/Z, fY/Z, f)^T$, both in $\mathbb{R}^3$ Euclidean coordinates, but with the second point on the image plane. The third coordinate of the second point can be ignored, because it will be the same for any point on the image plane, reducing the transformation to

$$(X, Y, Z)^T \mapsto (fX/Z, fY/Z)^T. \tag{3.2}$$

Since we are using homogeneous coordinates the point on the image plane can be written as $(fX, fY, Z)^T$. This means the projection $\mathbf{x} = \mathrm{P}\mathbf{X}$ can be written in matrix form as

$$\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{3.3}$$

The line perpendicular to the image plane that passes through $\mathbf{C}$ is called the principal axis. In **Figure** 3.1(b) this line corresponds to the Z-axis. The point where

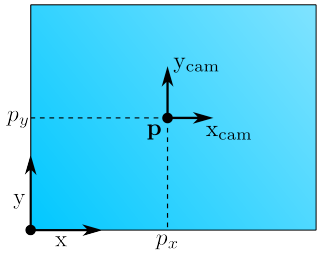


Figure 3.2: Image coordinates start at the bottom left and the principal point **p** usually lies near the middle of the image.

this line intersects the image plane is called the principal point. When working in image coordinates this point might not be at the origin. In **Figure** 3.2 the rectangle represents an image produced by a camera with the image coordinates measured from the bottom left. The principal point **p** can have any coordinates, but usually lies close to the centre of the image.

To compensate for the offset of the principal point in the projection we need to add the x and y coordinates of **p** to the x and y coordinates of the projected point such that

$$(X, Y, Z)^T \mapsto (fX/Z + p_x, fY/Z + p_y)^T. \tag{3.4}$$

The matrix representation of this transformation now changes to

$$\begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{3.5}$$

The projection $\mathbf{x} = \mathrm{P}\mathbf{X}$ can now be separated into the form

$$\mathbf{x} = \mathrm{K}\,[\mathrm{I} \mid \mathbf{0}]\,\mathbf{X}, \tag{3.6}$$

where

$$\mathrm{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

The matrix K is called the calibration matrix (or intrinsic parameter matrix) and contains information that defines the inner workings of the camera. The position

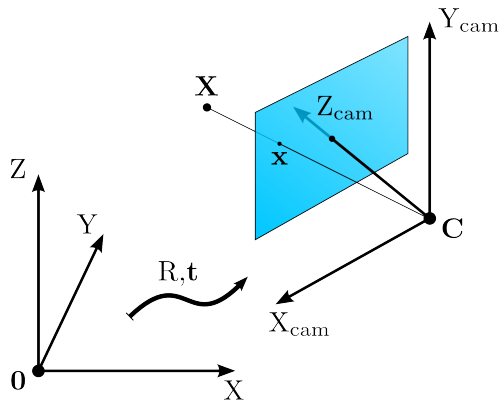


Figure 3.3: The world coordinate system is related to the camera coordinate system through a translation $\mathbf{t}$ and a rotation R.

and orientation of the camera in world coordinates are external parameters and are considered next.

In **Figure** 3.3 $\mathbf{X}$ now represents a point in world coordinates. For the projection to map this point to a point on the image plane, as in **Equation** 3.6, we first need to convert $\mathbf{X}$ to a point in the camera's coordinate system. The conversion is done by first subtracting $\tilde{\mathbf{C}}$ from $\tilde{\mathbf{X}}$, where $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{X}}$ denote the Euclidean versions of the homogeneous vectors, so that the origin of the new coordinate system is translated to $\tilde{\mathbf{C}}$. A rotation R is then used to align the coordinate system with the camera's orientation. This conversion can be written as

$$\tilde{\mathbf{X}}_{cam} = \mathrm{R}\left(\tilde{\mathbf{X}} - \tilde{\mathbf{C}}\right). \tag{3.8}$$

In homogeneous coordinates this becomes

$$\mathbf{X}_{cam} = \begin{bmatrix} \mathrm{R} & -\mathrm{R}\tilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{3.9}$$

The point $\mathbf{X}_{cam}$ is now in the same coordinate system that $\mathbf{X}$ was in **Equation** 3.6 so we can combine **Equations** 3.6 and 3.9 to get

$$\mathbf{x} = \mathrm{KR}\left[\mathrm{I}|-\tilde{\mathbf{C}}\right]\mathbf{X}. \tag{3.10}$$

The pinhole camera model just derived has 9 degrees of freedom: 3 for K (elements f, p_x and p_y), 3 for R (rotation about the 3 axes of rotation) and 3 for $\tilde{\mathbf{C}}$.

The model assumes that the projected coordinates have equal scales in both axes, which would require square pixels in a CCD camera. However, this is not always the case, so an extra degree of freedom is introduced to account for the ratio between the two axes. If the scale in the X-axis is m_x and the scale in the Y-axis is m_y then the matrix K is multiplied from the left by $diag(m_x, m_y, 1)$ changing the intrinsic parameter matrix to

$$\mathrm{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

Thus $\alpha_x = m_x f$ and $\alpha_y = m_y f$ represents the focal length in the x and y directions respectively and $x_0 = m_x p_x$ and $y_0 = m_y p_y$ the principal point's coordinates, all in pixel measurements.

For added generality a skew factor s can be introduced such that

$$\mathrm{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.12}$$

bringing the total number of degrees of freedom to 11, the same as for a 3×4 matrix defined up to an arbitrary scale.

In the next section we discuss how P is calculated for a real world camera.

3.2 Single camera calibration

To obtain the projection matrix P we need a set of point correspondences $\mathbf{X}_i \leftrightarrow \mathbf{x}_i$ such that $\mathbf{x}_i = \mathrm{P}\mathbf{X}_i$ for all correspondences i. From these point correspondences we need a set of linear equations to calculate the elements of P. Since P has 11 degrees of freedom we need at least 11 independent linear equations to calculate them uniquely.

To get these linear equations we write $\mathbf{x}_i = \mathrm{P}\mathbf{X}_i$ as the vector cross product $\mathbf{x}_i \times \mathrm{P}\mathbf{X}_i = \mathbf{0}$. If we write P as a vector of its rows then,

$$\mathrm{P}\mathbf{X}_i = \begin{pmatrix} \mathbf{p}_1^T \mathbf{X}_i \\ \mathbf{p}_2^T \mathbf{X}_i \\ \mathbf{p}_3^T \mathbf{X}_i \end{pmatrix}, \tag{3.13}$$

where $\mathbf{p}_j^T$ denotes the jth row. The cross product now becomes

$$\mathbf{x}_i \times \mathrm{P}\mathbf{X}_i = \begin{pmatrix} y_i\mathbf{p}_3^T\mathbf{X}_i - w_i\mathbf{p}_2^T\mathbf{X}_i \\ w_i\mathbf{p}_1^T\mathbf{X}_i - x_i\mathbf{p}_3^T\mathbf{X}_i \\ x_i\mathbf{p}_2^T\mathbf{X}_i - y_i\mathbf{p}_1^T\mathbf{X}_i \end{pmatrix}, \tag{3.14}$$

where $\mathbf{x}_i = (x_i, y_i, w_i)^T$. To rewrite this equation we can use the fact that $\mathbf{p}_j^T\mathbf{X}_i = \mathbf{X}_i^T\mathbf{p}_j$ and thus it becomes

$$\begin{bmatrix} \mathbf{0}^T & -w_i\mathbf{X}_i^T & y_i\mathbf{X}_i^T \\ w_i\mathbf{X}_i^T & \mathbf{0}^T & -x_i\mathbf{X}_i^T \\ -y_i\mathbf{X}_i^T & x_i\mathbf{X}_i^T & \mathbf{0}^T \end{bmatrix} \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix} = \mathbf{0}, \tag{3.15}$$

a set of 3 linear equations in the form $\mathrm{A}_i\mathbf{p} = \mathbf{0}$. A_i is a 3×12 matrix and $\mathbf{p}$ is a 12-vector containing all the elements of P where

$$\mathbf{p} = \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}, \quad \mathrm{P} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix} \tag{3.16}$$

and p_i is the i-th element of $\mathbf{p}$.

Of the three linear equations only the first two are linearly independent, since the third can be constructed by x_i times the first row plus y_i times the second. This means the set of equations is reduced to

$$\begin{bmatrix} \mathbf{0}^T & -w_i\mathbf{X}_i^T & y_i\mathbf{X}_i^T \\ w_i\mathbf{X}_i^T & \mathbf{0}^T & -x_i\mathbf{X}_i^T \end{bmatrix} \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix} = \mathbf{0}. \tag{3.17}$$

We see that one point correspondence yields two equations and since we need 11 to define P uniquely it is clear that $5\frac{1}{2}$ point correspondences are required for the calculation. The $\frac{1}{2}$ means that we require six point correspondences, but only one of the two equations produced by the sixth. However there is little point in ignoring any extra information as will become clear shortly.

The solution is obtained by stacking all 11 equations produced into a single matrix A and solving $\mathrm{A}\mathbf{p} = \mathbf{0}$. A is an 11×12 matrix and will, in theory, be of rank 11. The solution vector $\mathbf{p}$ is the 1-dimensional right null-space of A. However,

Figure 3.4: A picture of the chessboard pattern used for calibration. The zig-zag lines mark the positions of the corners detected by OpenCV.

the point correspondences might contain measurement errors resulting in a matrix A that has a rank less than 11. In this case A will not have a 1-dimensional right null-space.

To overcome this problem an over-determined system is used; if n is the number of point correspondences then $n \geq 6$. In this case A becomes a matrix with dimensions $2n \times 12$ and the solution to $\mathrm{A}\mathbf{p} = \mathbf{0}$ is approximated in a least-squares sense using singular value decomposition (SVD). If the SVD of A is such that $\mathrm{A} = \mathrm{UDV}^T$ then the solution vector $\mathbf{p}$ is the singular vector corresponding to the smallest singular value. So if the diagonal matrix D has positive entries arranged in descending order, then $\mathbf{p}$ is the last column of V.

The only remaining issue is that of obtaining the point correspondences. This is usually done by means of a calibration object of which the exact dimensions in world coordinates are known. The general convention is to use a chessboard pattern for this object [29]. The advantage of using this pattern is that the corners can be easily detected in an image, up to sub-pixel accuracy. Several pictures of this calibration object are taken at various angles and positions. The set of point correspondences is then constructed by the coordinates of the corners in the image and our knowledge of the object's dimensions.

We used OpenCV for our calibration, for more information see [30]. They provide functions for automatic corner detection, sub-pixel refinement and calibration. An example of one of the pictures taken of the calibration object is given in **Figure** 3.4. More can be seen in **Section** 3.4, where we deal with stereo calibration.

(a) (b)

Figure 3.5: An example of radial distortion taken from [31]. (a) The distorted image. Notice the bent edges of the ceiling and the door. (b) The undistorted image, where all the lines are straight.

3.3 Radial distortion

The pinhole model discussed in the previous section assumes that the rays of light hit the CCD in a linear fashion. In other words the feature **X**, camera center **C** and the point on the image plane **x** all lie on the same line. However, in commercially available cameras the lenses cause some distortion as can be seen in **Figure** 3.5 (a).

This type of distortion is called radial distortion and causes straight lines to bend. The degree to which a line is bent is proportional to its distance from some point in the image called the center of distortion. This point usually corresponds to the principal point of the image and thus lines on the edges of the image are the most distorted.

The distortion of any measured point (x_d, y_d) can be modelled by a function $L(\tilde{r})$ where $\tilde{r}$ is the distance from the undistorted point (x, y) to the center of distortion (x_c, y_c), summarised in the equation

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = L(\tilde{r}) \begin{pmatrix} x \\ y \end{pmatrix}. \tag{3.18}$$

To undo this warping we need to move each pixel to its undistorted position. This correction is written as

$$x = x_c + L(r)(x_d - x_c), \tag{3.19}$$

$$y = y_c + L(r)(y_d - y_c), \tag{3.20}$$

where r is the distance from the distorted point (x_d, y_d) to the center of distortion (x_c, y_c). If the axes are not of equal scale the equations have to be modified to take this into account.

The function $L(r)$ is approximated using a Taylor expansion $L(r) = 1 + \kappa_1 r + \kappa_2 r^2 + \kappa_3 r^3 + \ldots$ where the coefficients $\kappa_1, \kappa_2, \kappa_3, \ldots$ and the center of distortion x_c, y_c are considered internal parameters of the camera. They can be calculated, at the same time as the rest of the camera's internal parameters, by measuring the straightness of lines on the calibration object.

3.4 Stereo camera calibration

By definition a stereo system requires two cameras to work. For the second camera to be of any use we need to know its exact coordinates and orientation relative to the other one. To obtain this information we calibrate the two cameras individually, the same way we would a single camera, but not completely separate.

Instead of taking pictures of the calibration object for each camera separately, we take two pictures, one for each camera, of the calibration object simultaneously. This way we know the object will have exactly the same world coordinates. As long as the cameras' coordinates are measured relative to the same world coordinates we will be able to determine their spatial relationship.

We used for example a total of 41 image pairs for our calibration, some of which are shown in **Figure** 3.6. From here we use OpenCV to automatically calculate the intrinsic and extrinsic parameters as well as the distortion coefficients. The resulting values for K, R and **C** of the two camera matrices are

$$\mathrm{K}_1 = \begin{bmatrix} 1022.05 & 0.0 & 282.23 \\ 0 & 1025.7 & 197.63 \\ 0 & 0 & 1.0 \end{bmatrix}, \tag{3.21}$$

$$\mathrm{K}_2 = \begin{bmatrix} 1015.9 & 0.0 & 294.57 \\ 0 & 1017.57 & 171.96 \\ 0 & 0 & 1.0 \end{bmatrix}, \tag{3.22}$$

$$\mathrm{R}_1 = \begin{bmatrix} -0.01 & 0.99 & 0.15 \\ 0.95 & -0.04 & 0.3 \\ 0.3 & 0.15 & -0.94 \end{bmatrix}, \tag{3.23}$$

$$\mathrm{R}_2 = \begin{bmatrix} 0.02 & 0.99 & 0.13 \\ 0.95 & -0.06 & 0.3 \\ 0.31 & 0.12 & -0.94 \end{bmatrix}, \tag{3.24}$$

$$\mathbf{C}_1 = \begin{pmatrix} -17.23 \\ -4.92 \\ 66.75 \end{pmatrix}, \tag{3.25}$$

$$\mathbf{C}_2 = \begin{pmatrix} -9.97 \\ -4.2 \\ 66.76 \end{pmatrix}, \tag{3.26}$$

where the values where all rounded to two decimal places[1]. Different R and $\mathbf{C}$ values are obtained for every calibration image, relative to the calibration object, so we only show that of the first image.

From the K and R matrices it is clear that the cameras have almost the same intrinsic parameters and external orientation, which is desirable for stereo as we will see in the next section. The baseline of our stereo rig (see **Section** 5.1) is roughly 10 cm. So the Euclidean distance between the two camera centers, $\mathbf{C}_1$ and $\mathbf{C}_2$, should be the same. The coordinates of the camera centers are scaled according to the size of the squares on the chessboard. During the calibration the length of the side of a square is assumed to be 2 units of an arbitrary measure. In real world coordinates its length is 2.75 cm, so if the distance between the two centers is

$$\sqrt{(-17.23 + 9.97)^2 + (-4.92 + 4.2)^2 + (66.75 - 66.76)^2} = 7.3, \tag{3.27}$$

we have to scale the value by a factor of $\frac{2.75}{2}$ to get the real world distance, yielding

$$7.3 \times \frac{2.75}{2} = 10.04 \text{ cm}, \tag{3.28}$$

which suggests that the calibration was a relative success.

[1] This is a bad idea in an actual implementation, since small errors at the camera can lead to large errors a bit further away.

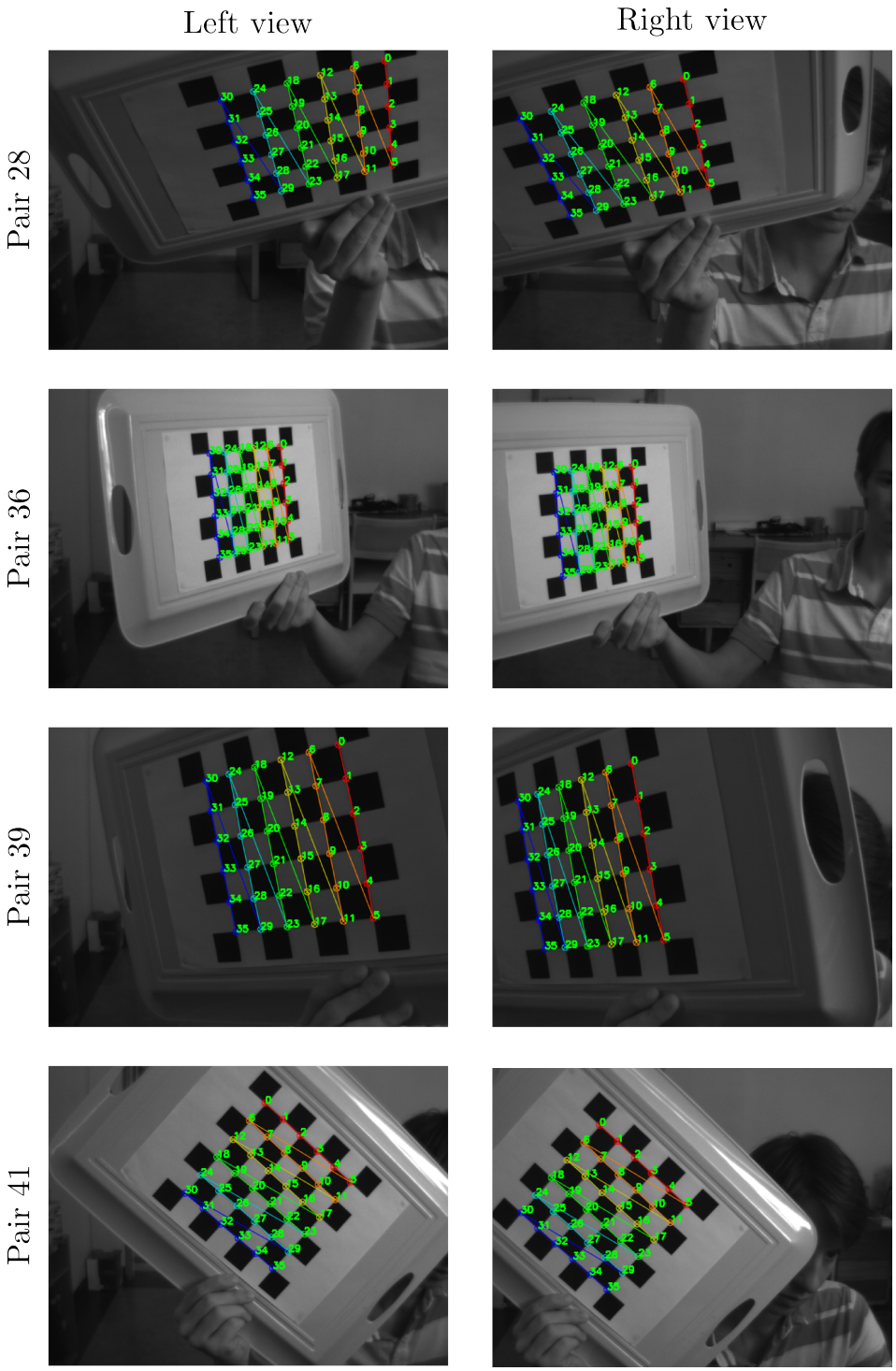


Figure 3.6: Examples of the image pairs used for calibration. Corners are detected using OpenCV and numbered 0 to 35. The left and right pictures are taken at exactly the same time, so corners with the same number in both images correspond to exactly the same point in world coordinates.

3.5 Epipolar geometry

Up to this point we have covered the geometry of a single camera and introduced a second camera to form a stereo system. We now analyse the relationship between these two cameras to see how they can be used to eventually triangulate $3D$ coordinates.

Figure 3.7 depicts a general stereo system. $\mathbf{C}_1$ and $\mathbf{C}_2$ are the camera centers in world coordinates, L and R are the image planes of the left and right cameras, $\mathbf{x}_1$ and $\mathbf{x}_2$ are the corresponding points of the same feature in their respective image planes and $\mathbf{X}$ is the $3D$ position of that feature. If we assume camera matrices of the form

$$\mathrm{P}_1 = \mathrm{K}_1\mathrm{R}_1\left[\mathrm{I}|-\tilde{\mathbf{C}}_1\right], \tag{3.29}$$

$$\mathrm{P}_2 = \mathrm{K}_2\mathrm{R}_2\left[\mathrm{I}|-\tilde{\mathbf{C}}_2\right], \tag{3.30}$$

as discussed in **Section** 3.1, then we know

$$\mathbf{x}_1 = P_1\mathbf{X}, \tag{3.31}$$

$$\mathbf{x}_2 = P_2\mathbf{X}. \tag{3.32}$$

Since our aim is to reconstruct a $3D$ model from the images we need to work in the opposite direction from $\mathbf{x}_1$ and $\mathbf{x}_2$ in order to triangulate $\mathbf{X}$. This triangulation is straightforward. The difficulty of stereo-vision is finding for a given point $\mathbf{x}_1$ its corresponding point $\mathbf{x}_2$.

For a dense reconstruction we need to find corresponding points for every pixel in one image. If L is the reference image and $\mathbf{x}_1$ and $\mathbf{x}_2$ are measured in pixel coordinates, then for every pixel $\mathbf{x}_1$ in L we need to search through every possible pixel in R to find a 'best' match, which of course implies a horrendous amount of computation. Luckily the search space can be reduced drastically by the use of epipolar geometry.

In epipolar geometry the points where the baseline $\mathbf{C}_1$ to $\mathbf{C}_2$ intersects the image planes are called the epipoles, $\mathbf{e}_1$ and $\mathbf{e}_2$ in **Figure** 3.8. The plane defined by the ray from $\mathbf{C}_1$ through $\mathbf{x}_1$ (and eventually through $\mathbf{X}$) and the line from $\mathbf{C}_1$ to $\mathbf{C}_2$,

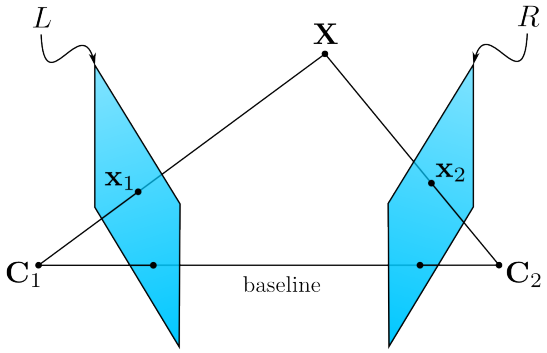


Figure 3.7: General stereo setup of two cameras. The left image plane L corresponds to camera $\mathbf{C}_1$ and the right image plane R corresponds to camera $\mathbf{C}_2$. The baseline connects the two camera centers and $\mathbf{X}$ projects onto the two image planes at $\mathbf{x}_1$ and $\mathbf{x}_2$.

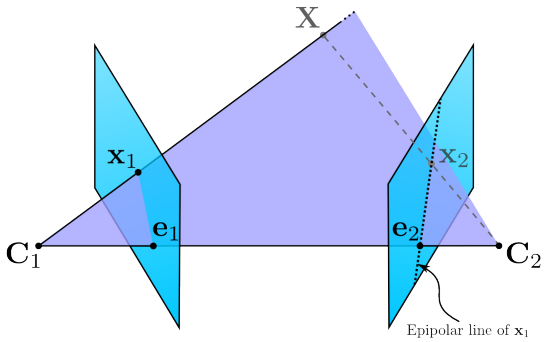


Figure 3.8: The epipolar plane, defined by the baseline and $\mathbf{x}_1$ on the reference image L, cuts through R to form an epipolar line. The points where the baseline intersects the image planes, $\mathbf{e}_1$ and $\mathbf{e}_2$, are called the epipoles.

called an epipolar plane, cuts through R in a single line, called an epipolar line. Finding $\mathbf{x}_2$ is thus limited to a search along the epipolar line.

Usually one would calculate the fundamental matrix F of the two cameras at this stage. The fundamental matrix is used to find the epipolar line $\mathbf{l}$ in the corresponding image for any point $\mathbf{x}$ with the equation

$$\mathbf{l} = \mathrm{F}\mathbf{x}. \tag{3.33}$$

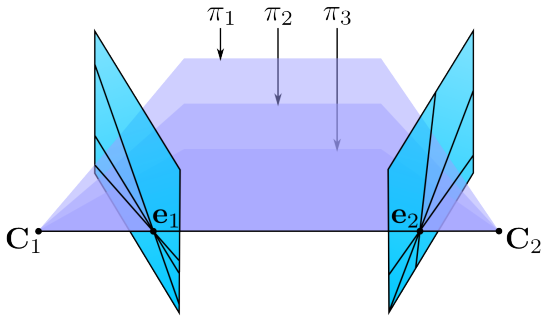


Figure 3.9: Planes $\pi_{1,2,3}$ cut through the image planes to form three epipolar lines. The set of planes that rotate around the baseline form the epipolar pencil.

However, we will be simplifying the geometry even further, to skip this step. Before we can do that we need to look at epipolar geometry in more detail.

Any epipolar plane defined by any point $\mathbf{x}$ in our reference image will always contain the baseline. This means that they are all related by a rotation around the baseline as shown in **Figure** 3.9. The epipolar lines of these planes all cross at the epipolar points and the pattern resembles the tip of a pencil. Therefore the set of all possible epipolar planes is called the epipolar pencil.

If we transform the images so they are both on the same plane and parallel to the baseline we end up with the situation in **Figure** 3.10. Since the baseline is parallel to the images it never intersects them, hence there are no epipoles. And in turn, no epipoles imply that the epipolar lines never cross one another. In other words the epipolar lines are parallel.

If the x-axis of the images are aligned with the baseline it means that each epipolar line will correspond to a single scanline. So if we are trying to find a match for a point with a certain y-coordinate we only need to search the scanline in the corresponding image with the same y-coordinate! The process of transforming the images into this state is called rectification and discussed in the next section.

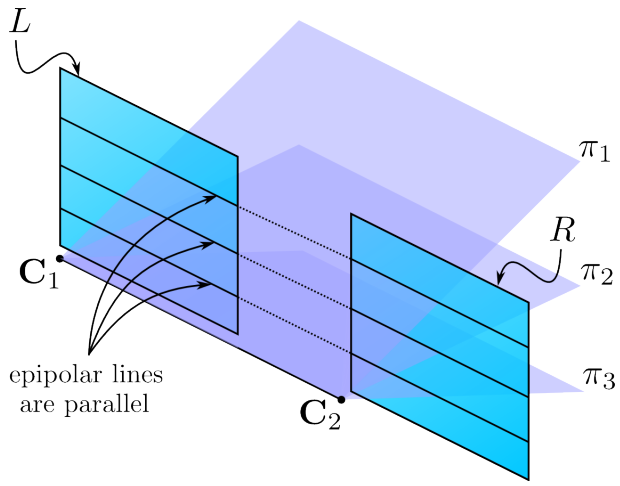


Figure 3.10: If the images are on the same plane and parallel to the baseline the epipolar lines become parallel.

3.6 Image rectification

Image rectification builds upon the search constraints prescribed by epipolar geometry by attempting to projectively transform the images in such a way that the epipolar lines are perfectly parallel and horizontal. This would narrow the search for a correspondence down to a single scanline. Transforming such that the epipolar lines are parallel will require that the two image planes are coplanar. This means we need to define a new rotation R_n and intrinsic parameter matrix K_n to which the images will be transformed.

The new intrinsic parameters can be chosen arbitrarily, but a simple choice would be

$$\mathrm{K}_n = \left(\mathrm{K}_1 + \mathrm{K}_2\right)/2, \tag{3.34}$$

if the camera matrices P_1 and P_2 are defined as before. For the images to be coplanar the Z-axis of the new orientation defined by R_n has to be perpendicular to the baseline (the line between $\mathbf{C}_1$ and $\mathbf{C}_2$). However, this does not insure that the scanlines of the images are aligned. To align them they must be parallel to the baseline. So we choose our first unit vector $\mathbf{r}_1$, corresponding to the X-axis of the new orientation, along the baseline and it becomes

$$\mathbf{r}_1 = \frac{(\tilde{\mathbf{C}}_1 - \tilde{\mathbf{C}}_2)}{\left\|\tilde{\mathbf{C}}_1 - \tilde{\mathbf{C}}_2\right\|}. \tag{3.35}$$

The new Y-axis must be orthogonal to $\mathbf{r}_1$, but is otherwise unconstrained. So we choose

$$\mathbf{r}_2 = \mathbf{k} \times \mathbf{r}_1, \tag{3.36}$$

where $\mathbf{k}$ is the unit vector of the principal ray of the old reference image. This helps to minimise the distortion caused by the re-projection[2]. The final unit vector $\mathbf{r}_3$, corresponding to the Z-axis, must be orthogonal to both $\mathbf{r}_1$ and $\mathbf{r}_2$ so it simply becomes

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2. \tag{3.37}$$

The new rotation R_n can now be written as

$$\mathrm{R}_n = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \mathbf{r}_3^T \end{bmatrix}. \tag{3.38}$$

To remap the images we define the transformations T_1 and T_2, such that

$$\mathrm{T}_1 = \mathrm{M}_n\mathrm{M}_1^{-1}, \quad \mathrm{T}_2 = \mathrm{M}_n\mathrm{M}_2^{-1}, \tag{3.39}$$

where $\mathrm{M}_n = \mathrm{R}_n\mathrm{K}_n$ and $\mathrm{M}_i = \mathrm{R}_i\mathrm{K}_i$ for $i = 1, 2$. Any points $\mathbf{x}_1$ and $\mathbf{x}_2$ on the original images are now mapped according to

$$\mathbf{x}_1' = \mathrm{T}_1\mathbf{x}_1, \quad \mathbf{x}_2' = \mathrm{T}_1\mathbf{x}_2, \tag{3.40}$$

where $\mathbf{x}_1'$ and $\mathbf{x}_2'$ are points on the rectified images. **Figure** 3.11 illustrates. In essence this transformation back-projects all the points on the images of both cameras and then re-projects them using the same orientation and intrinsic parameters. This effectively makes the two cameras exactly the same, but translated along the baseline.

The next issue to consider is the possibility that there might not be a true match for every pixel in the reference image. A certain feature in the reference image could be obscured, or occluded, from the view of one of the cameras. This topic is covered in the next section.

[2]In an attempt to limit the distortion that a projective transformation causes in the first place, one should try to setup the cameras so that they are both more or less, and as closely as possible, facing the same direction perpendicular to the baseline.

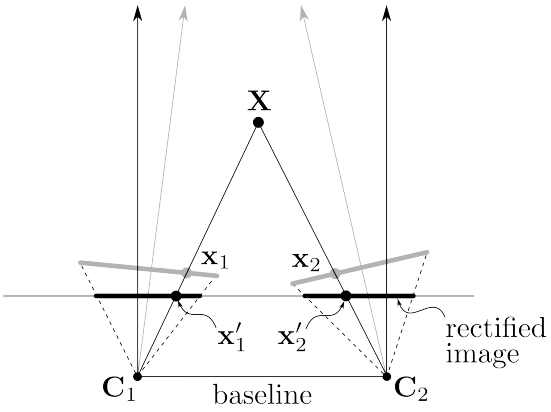


Figure 3.11: Image rectification maps both images onto the same plane parallel to the baseline.

3.7 Occlusions and disparities

The phenomenon that occurs when some object or feature is visible in one image but not in the other is called an occlusion. **Figure** 3.12 gives an example of a view through a slit or gap. Occluded areas are marked in gray. Left-occlusion areas, i.e. areas not visible in the left camera view, are denoted by L_{occ} and right-occlusion areas, i.e. areas not visible in the right camera view, are denoted by R_{occ}.

Figure 3.12 also illustrates what is meant by disparity. The pixel $\mathbf{x}_2^*$ has the same coordinates as $\mathbf{x}_2$, but in the second image. If $\mathbf{x}_2'$ represents the same feature then the distance from $\mathbf{x}_2^*$ to $\mathbf{x}_2'$ is the disparity associated with $\mathbf{x}_2$. Therefore the disparity of a pixel represents the distance that the pixel has moved from one image to the other. Intuitively, the larger the disparity the closer that feature is to the cameras.

3.8 Triangulation

After all the effort we have been through to make our stereo-correspondence problem easier, finding the actual $3D$ coordinates of a point becomes fairly straightforward, assuming that we have a situation where the images have been undistorted and rectified. Further assuming that we have found the image coordinates $\mathbf{x}_L$ and $\mathbf{x}_R$, of a feature $\mathbf{X}$, in the left and right images respectively, we can calculated $\mathbf{X}$ by similar triangles.

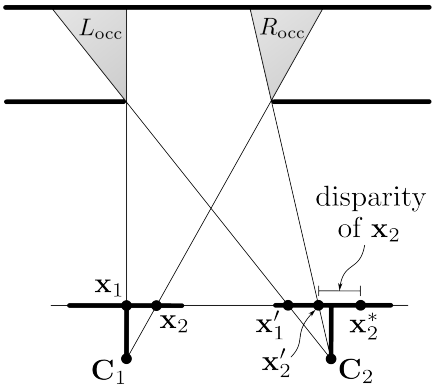
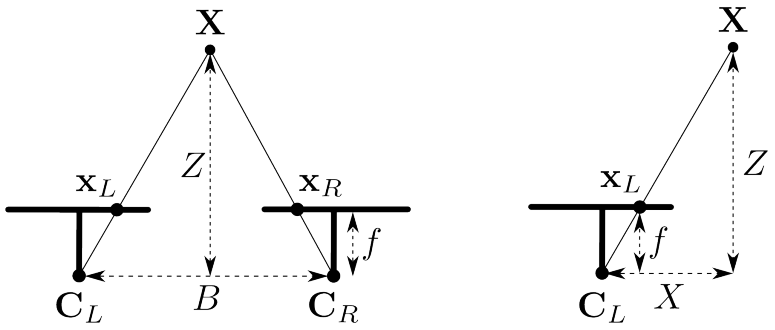


Figure 3.12: Examples of left- and right-occlusion areas (shaded). The disparity associated with pixel $\mathbf{x}_2$ is marked as the distance from $\mathbf{x}_2^*$ to $\mathbf{x}_2'$.

$d = x_L - x_R$



(a) (b)

Figure 3.13: Triangulation of $\mathbf{X}$ using similar triangles. (a) Triangulation of the depth coordinate Z using the disparity d of the two image coordinates of $\mathbf{X}$. (b) Calculation of the X coordinate of $\mathbf{X}$ using ratios of triangles. Y is calculated in the same way.

If the disparity is defined by $d = x_L - x_R$, the length of the baseline is B and the focal length is f then by referring the **Figure** 3.13(a), we have

$$\frac{B-d}{Z-f} = \frac{B}{Z} \;\Rightarrow\; Z = \frac{fB}{d}, \tag{3.41}$$

where Z is the depth coordinate of $\mathbf{X}$.

From the formula for Z it is clear that the distance from the cameras is inversely proportional to d, as mention in the previous section. In Figure 3.14 we plot disparity versus distance. We also plot the distance between two consecutive integer values of d. It is clear that for small disparities accurate measures become nearly impossible.

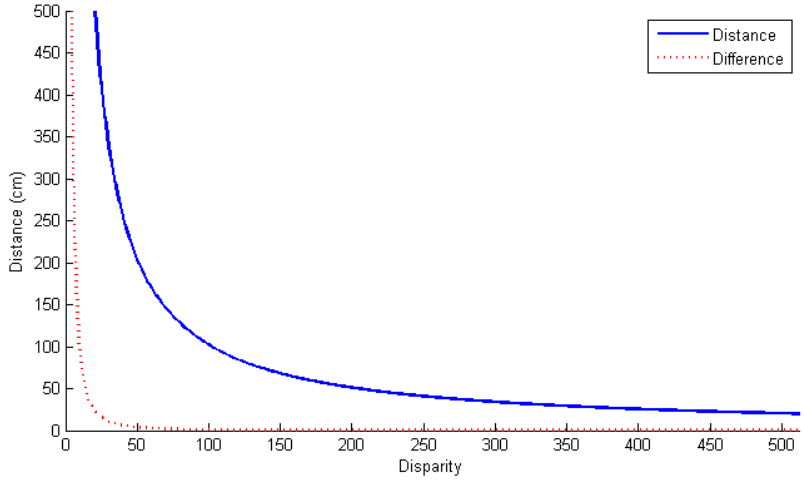


Figure 3.14: Disparity versus distance. This graph assumes that the focal length f is 1020 (measured in pixels) and the baseline B of the stereo rig is 10 cm. 'Distance' marks the distance from the cameras given a certain disparity. 'Difference' marks the distance between the current disparity and the next. A mistake in the lower ranges of disparities will be much more severe than in the higher ranges.

If we choose the left camera as our reference image, calculating the X coordinate of $\mathbf{X}$ is also simple, see **Figure** 3.13(b). By using similar triangles, again, we have

$$X = \frac{(x_L - p_x)}{f} Z, \tag{3.42}$$

where p_x is the horizontal offset of the principal point. It has to be subtracted if $\mathbf{x}_L$ is in image coordinates. To write this in terms of B and d we substitute the formula for Z and get

$$X = \frac{(x_L - p_x)}{f} \frac{fB}{d} \quad \Rightarrow \quad X = \frac{x_L B}{d}. \tag{3.43}$$

Applying the same technique to get Y.

The whole process can now be neatly packed into a single matrix multiplication using homogeneous coordinates. If we define the matrix Q such that

$$\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \mathrm{Q} \begin{pmatrix} x \\ y \\ d \\ 1 \end{pmatrix}, \tag{3.44}$$

then

$$\mathrm{Q} = \begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{1}{B} & 0 \end{bmatrix}. \tag{3.45}$$

If we have a disparity value for every pixel in the reference we can now transform them into a dense cloud of $3D$ coordinates. Obtaining these disparities is the subject of the next chapter. For more information on calibration, rectification and triangulation refer to [30] and [32].