Stereoscopy and Scene Reconstruction

M.J.P. Hagenaars[1]

June 20, 2011

BACHELOR THESIS

Artificial Intelligence
Radboud University Nijmegen

Supervised by dr. F.A. Grootjen and dr. L.G. Vuurpijl

[1]mhagenaars@student.ru.nl

Abstract

Depth perception is of great use to humans in many situations, therefore it could be very useful for artificial systems to be able to use this capability in their many applications. One of the most common means of deriving depth information from two images is stereoscopy, or stereopsis. In this bachelor thesis project a novel approach to computational stereoscopy is introduced in the form of a two-stage model. First a very fast, but inaccurate form of template matching ("Sloppy Matching") is performed, which is then improved upon by an iterative quasi-physical simulation to remove erroneous matches ("Corrective Smoothing").

Contents

Chapter 1

Introduction

There are many situations in which problems arise with tasks that are either too tedious or dangerous for most human beings to perform. In such cases it can often be useful to deploy some piece of equipment and have it do large portions of the work for us. One can see, for example, the dangers to human rescue workers when entering an unstable building looking for survivors of an accident. Especially when an explosion may be imminent or toxic gases have been released.

In situations like this, dangers can potentially be avoided if detailed information about the site of the accident is available to the rescue workers in a very early stage. One can think of an autonomous robot that can be sent in ahead of any human rescuers, to investigate the situation without putting any more people in harm's way. It should collect relevant information like structural integrity of buildings or machinery, locations and conditions of victims, and so on.

A video feed from such a non-, semi-, or fully autonomous "camera cart" can be a great improvement to the situational awareness of the rescue workers. It would be even better, however, if the robot could generate a full-fledged three dimensional representation of the site of the accident, including aforementioned points of interest like victim positions.

Analyzing three-dimensional aspects of some "world" requires methods and instruments that facilitate measuring the relevant properties in that world. Bats and dolphins, for example, are known to use echolocation in navigation and foraging behavior. Furthermore, many sea creatures have organs called lateral lines, that enable them to sense movement in their surroundings. Most people, however, tend to associate three-dimensional "world sensing" with the principle of perceiving depth using two eyes.

Studies concerning these natural world sensing mechanisms are manifold. They have inspired researchers to develop artificial systems that allow for similar capabilities in technological systems. Think of microphones, radar / sonar, cameras, lasers (for measuring distances), and so on. Since humans and many animals are capable of extracting rather detailed information about their surroundings using their optical organs, it makes sense to investigate these capabilities for use in the development of autonomous systems like the robot scout mentioned before.

Other applications that are related to depth perception can be found in the entertain-

ment industry. As they are becoming better and more affordable, television sets that allow for displaying specially recorded, three-dimensional imagery [MP04] are gaining popularity all over the world. Some companies are developing game controllers that use three-dimensional measurements as input data [LTG+10]. A completely different application is that of ground surface topography, which analyzes satellite imagery to form Digital Elevation Models (DEMs) of the earth [HWL03]. The diversity of research and applications related to depth perception indicate that this currently is a rather hot topic, that is well-worth investigating.

1.1 Stereoscopy

This bachelor thesis project focuses primarily on depth perception in artificial systems through the use of stereoscopy, inspired by binocular disparity in humans [Qia97]. Stereoscopy, or stereopsis, is the ability to blend two slightly different views of a scene together to allow for judging depth and distances [PP84]. The principle of stereopsis is based on deriving parallax information from separate two dimensional images, captured by a pair of cameras or eyes positioned at a relatively small distance from each other. Parallax is the relative distance between an object's two-dimensional projections on two viewports (retina, or camera film or sensor) from a slightly different perspective. The term binocular disparity refers to the horizontal parallax caused by the eyes being horizontally positioned at a distance from each other. To detect disparities, a part of the brain — the visual cortex — is able to match features and objects, that are seen with one eye, to what is seen by the other eye [PP84].

Throughout this paper, the terms stereoscopy and stereopsis are often abbreviated to "stereo", especially when used in conjunction with related terminology. This is to prevent unnecessary verbosity, because in this paper, like in much of the related literature, the word "stereo" is not used in any other context.

Humans, as well as many other animals, use binocular disparity cues to derive depth information from their surroundings. This allows for estimating distances in the world, determining which elements are closer than others, and forming an internal representation of the world. Skills like these are of great advantage to their wielders and, as such, people have been trying to replicate the principles of depth perception to many ends.

These imitations are usually based on a more abstracted view of depth perception, considering the eyes as pinhole camera models. In reality, each eye captures a two-dimensional, but curved and upside down, projection of a visible portion of the world on its retina, whereas in a pinhole camera model, as it is used in this thesis, the projection is two-dimensional as well as it is perfectly flat and upright. Also, a pinhole camera model abstracts from many complicating factors, like lens distortions.

Besides through stereoscopy, humans can obtain depth cues from their surroundings by means of other mechanisms. There are other binocular depth cues, like convergence [BVD98], which has to do with muscle contractions around the eye balls that attempt to keep both eyes on the same object at some distance. Monocular depth cues include, for

example, perspective, occlusion, and motion parallax [RG82].

So there is a three-dimensional world (only counting spatial dimensions, but not time and such) that casts two-dimensional projections into a pair of eyes. Photosensitive ganglion cells collect this projected light so that the brain receives an image from each eye. Analyzing the differences between the two dimensional images is one of the ways the brain can infer depth information about the three-dimensional world. Although it is hard to achieve a high level of accuracy using this technique, useful models for converting a pair of two dimensional images into a partial, three-dimensional reconstruction of the world, are possible.

1.2 Computational stereo

In search of computational models for stereo vision, there are a few primary problems that will arise and must be solved at certain points in the process. Some of these problems will be hardware-related, while others may be of conceptual nature or about the software implementation. To allow for more specific research goals, not all aspects of stereo vision can be addressed in equal measure in the course of this project. Certain abstractions will be formulated to accomplish this.

In the case of this thesis project the focus is not so much on the hardware as it is on the algorithms used to attain stereoscopic capabilities for a computerized system. Therefore, abstractions will be made from the camera models, and even from the world that is going to be reconstructed, in order to provide a suitable environment in which to develop the desirable algorithms.

When all unnecessary and unrelated aspects to the model are eliminated, it becomes possible to formulate an approach to analyze the images that are fed into this part of the model. The way in which both images differ from each other unveils more spatial information about the perceived world than each of those images would on its own. By detecting and analyzing these differences, these added spatial properties can, to some extent, be derived.

Finally, a single representation can be compiled that encompasses all spatial information derived from the images, thereby implicitly converting the two dimensional images into an estimation of the viewed part of the three dimensional world. This can be seen as a building block for a more comprehensive internal representation of the world.

1.2.1 Epipolar geometry

One of the first issues in developing a computational stereo implementation is the matter of calibration, which deals with the physical geometry of the cameras or eyes. There are matters of external geometry, such as the relative positions and orientations of the cameras, and matters of internal geometry, like focal lengths, optical centers, and lens distortions [BBH03]. Although the use of the pinhole camera model makes many principles a lot easier to deal with, there still are many things that should be taken into account.

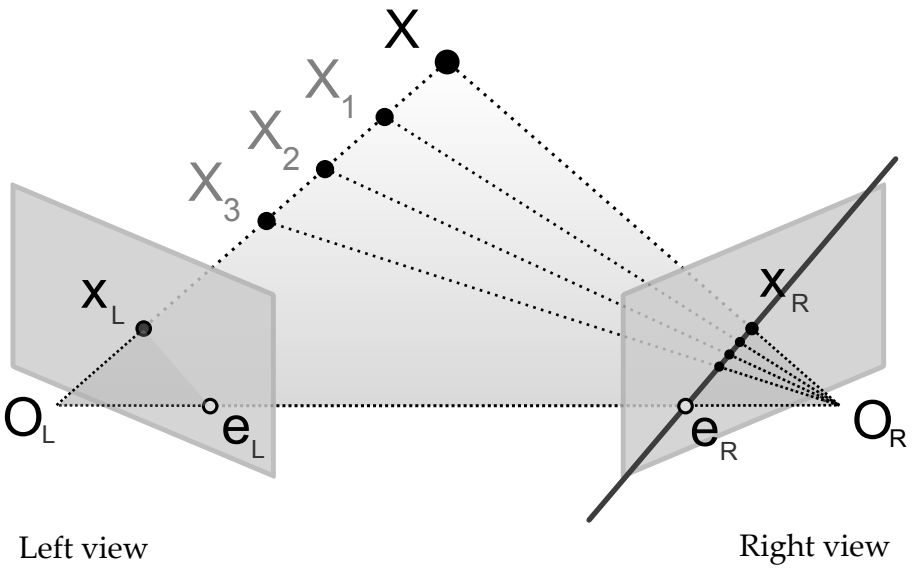


Figure 1.1: Example of epipolar geometry [Nor07].

Epipolar geometry [FLP01], which is the geometry of stereo vision, deals with how cameras at different positions capture the world differently. It describes the relations between three-dimensional points and how they are projected as two-dimensional points in two or more pinhole cameras. The name is derived from epipoles — the points that are the projection of a camera's focal point onto another camera's image plane. The plane formed by these epipoles and a point in the world is called an epipolar plane, and the intersection of an epipolar plane with a camera's image plane is called an epipolar line.

When all geometric properties of two cameras are known (*Figure 1.1*), and the projection X_L of a point X on the left camera's image plane is given, the epipolar line on the right camera's image plane is known as well. This is the epipolar constraint that holds for all corresponding points in two (or more) cameras. The projected point X_R in the other image must be located on the newly found epipolar line, because that satisfies the epipolar constraint as long as X is unknown. The problem of determining where X_R must be located on the epipolar line is called the correspondence problem. Now, if both projections, X_L and X_R, of point X are known, the intersection of the projection lines is the location of point X. This method of reconstructing three-dimensional points from their two dimensional projections is called triangulation.

1.2.2 Discrete images

Because of the finite digital format in which images are processed in computational systems, not all projections of points in the world, of which there is a theoretically infinite number, can be stored or displayed. Therefore a finite number of projections is estimated

by gathering color information for each pixel in the image. Without going into too much detail regarding the physics of light, this process can simply be thought of as averaging the color values of every visible point in a volume projected by each pixel. This volume is spanned by rays emanating from the camera's focal point and going through the corner points of the pixel. In photographs the horizontal disparity needed for stereo vision software causes some problems in the stereo images. Sensor noise, lens flares, lens distortions, and other imperfections are different in each photograph, but not because of stereo effects. These effects should therefore be eliminated wherever possible.

Whereas the color value for every pixel in real photography could be seen as an average over all points in the volume cast by each pixel, this principle is somewhat different in computer generated images. Not all points and the light they emit and reflect can be computed in a finite amount of time. Therefore the color values for each pixel are usually determined by a single point in the virtual world. This can be seen as a ray that is cast from the virtual camera's focal point, through the center of each pixel, up to the first visible point in the world. The color value for this pixel is determined by the material and the amount of light this point reflects. To smoothen out the images and improve visual quality certain post-processing effects, such as anti-aliasing [Dou88] and anisotropic filtering [GG99], are often applied. This can also have an effect on finding a viable solution to the correspondence problem.

1.2.3 Smooth surfaces

Another issue regarding the correspondence problem is that of large and smooth surfaces, such as blank walls, or overexposed areas. Due to the digital nature of these images it is possible that some details are too subtle or too small to be expressed in a change of color to some pixels. Solutions to these problems can be found in the areas of tone mapping [KMS05] in case of digital photography, or high dynamic range rendering [MGM06] in case of computer generated imagery. Depending on the resolution and overall quality of the image, some textured areas may appear completely monochromatic.

When these blank areas are surrounded by pixels that are more visually distinct and at different depths, it may be unclear to which depth level the blank pixels belong. An overexposed surface at an angle, for example, can easily be mistaken to be perpendicular to the viewer, unless the context of such an area can somehow be taken into account appropriately.

1.3 Previous research

Upto this chapter only general issues regarding stereoscopy are discussed. Next, the distinction between different categories of stereo algorithms is discussed, along with a brief elaboration on the history of research in the areas of (mostly computational) stereoscopy.

In 1838 Wheatstone invented the stereoscope [Whe38], causing a wealth of new research to emerge on the topic of binocular vision. More than one-and-a-half centuries

later, Lane and Thacker [LT94] introduced a categorization of computational stereo research. They address the great diversity of algorithms spawned by the inspiring principles behind stereoscopy by distinguishing between area based and feature based algorithms, as discussed in *1.3.1* and *1.3.2*. Some approaches, however, try to employ a synergetic combination of principles from both types of algorithms.

Throughout the literature researchers have had a profound interest in autonomous robot navigation. From rudimentary stereo in ground robotics [KTB89] to more advanced aerial systems [HSC+05], stereoscopy has proved a useful means of world sensing.

Dhond et al. [DA02] mention early works on "computational stereo for extraction of three-dimensional scene structure" in the 1970s and 80s that focus on the fundamental principles of stereo reconstruction and performance criteria.

From the early 1990s on, more progress was made in research regarding more specific problems, although general stereo research continued [K+93]. Some research focused on early occlusion detection, while others introduced transparency handling and real-time implementations. Active and dynamic stereo vision combines binocular disparity with optical flow in robot navigation.

Upto the early 2000s, research has advanced with new techniques for area and feature based matching, and methods for dealing with occlusion [BBH03]. Some projects added more cameras to achieve multi-camera stereo, while others focused on motion stereo with a single camera.

Much of the more recent research, apart from continuations of, and advances beyond, earlier work, is dedicated to practical applications of computational stereo in machine vision. Advances are made in autonomous obstacle detection and autonomous navigation. Examples are a robotic system for three-dimensional object search [ST10], and a system that can recognize free parking spaces [SJBK08].

1.3.1 Area based stereo

This category of stereo algorithms is characterized by its application in dense stereo matching – which is separately estimating depth for each pixel. It is based on the idea of pixel blocks that can be compared to each other using some measure of (dis)similarity. These measures often are based on normalized cross-correlation, or similar methods such as sums of squared differences or sums of absolute valued differences, which are computationally simpler.

Another branch of area based methods is that of least-squares region growing [OC89], where the matches (see *1.5.1*) are improved upon by using smoothness assumptions. Algorithms from yet another area based branch are seeking a solution to the correspondence problem by approaching a minimal cost mapping through simulated annealing [SB95].

1.3.2 Feature based stereo

Most feature based stereo algorithms provide for sparse stereo matching [Tay08]. That is, these algorithms perform stereo matching with image features, by which they can be classified. Beneficial to these approaches are the well-known statistical principles that support the properties of feature based representations. Features, while otherwise largely undefined, attempt to assign some meaning to the raw pixel data that can be used to unveil the depth information in the image pair. Features like edges and corners that belong to separate objects may hint at depth discontinuities.

The feature based approaches can roughly be divided into the following categories. Edge-string based algorithms assume the world consists of edges, only varying slightly between images, that separate objects [MF01]. As many man-made environments will contain straight edges that come together in corners [HS88], there are corner based algorithms to analyze these edge-linking elements to construct higher level edge structures. Texture region based algorithms [MMF91] split an image into blocks that can be roughly matched against its stereoscopic counterpart. Global surface continuity and local smoothness are then enforced through extensive smoothing processes.

1.4 Research question

The issues of calibration (*1.2.1*) and correspondence (*1.2.2* and *1.2.3*), that arise when dealing with stereoscopy, lead to the following research question:

> Is it possible to derive a depth map from two images by analyzing (horizontal) parallax, using a computationally flexible algorithm, so that it can be used for scene reconstruction?

This main question fits the characteristics of the underlying problems quite well, as it emphasizes the many possible approaches and (partial) solutions to these problems that researchers have come up with. In the process of finding an answer to the research question, a number of subquestions need to be addressed. These subquestions can be roughly divided into the following categories, of which the first two are discussed in-depth:

Calibration: What are the relative positions of the cameras in the world and their further geometric properties? These must be known, computable, or estimable in order to disclose the camera's epipolar properties. This allows for simplifying the model and reducing the correspondence problem to a one dimensional search problem.

Correspondence: One point in a three-dimensional space corresponds to a different point in each image plane. How can this correspondence be derived from only the two projected points? Indeed, how can this correspondence be derived even from just the pixels, that are estimations of the real points, in the images?

Reconstruction: Having derived all depth information from the images, what is a viable way to assemble a reconstruction of the original scene?

1.5 Two-stage model

In order to find a solution to the aforementioned problems a two-stage model is composed, mainly addressing the correspondence problem. Since the subject of computational stereo in general is too elaborate to be treated comprehensively in the course of this bachelor thesis project, the model will be delimited by certain assumptions and abstractions.

The matter of calibration is discussed briefly as it can be addressed in a relatively simple way by assuming the cameras to be positioned parallel to each other. A more elaborated solution to the correspondence problem is proposed, and a reconstruction example is given to demonstrate the model's effectiveness.

The basic idea behind the two stages in this model is to take a fast but sloppy estimate, and then correct it. The first stage estimates the depth information for every pixel in a very fast, but sloppy way. The second stage then corrects and smoothens out the result, by making use of certain constraints and probabilities. The workflow of this model is illustrated in *Figure 2.1* on page 10.

Both cameras will be positioned slightly apart, and in such a way that they are parallel to each other. Because the epipolar lines are then perfectly horizontal, all corresponding points will be at the same vertical offset in both images. This means that in seeking the corresponding point from one image, only one scanline in the other image is of interest. The correspondence problem is thereby reduced to a one dimensional search problem.

1.5.1 Sloppy Matching

The first stage in the proposed model is a simplified form of template matching [Bru09]. This process encompasses the search for occurrences of a small image in a larger image. To accomplish this, groups of pixels are compared to each other by using a (dis)similarity measure. Because of the assumed calibration settings, template matching only has to be done along horizontal lines. Also, a sizable portion of each scanline can be pruned, because the stereo analysis is oriented to one image in comparison to the other (see *2.2.2*).

As very little information about a pixel's context in the image is used in the computations in this first stage, because it is either unknown, or too computationally expensive, many erroneous matches may occur.

1.5.2 Corrective smoothing

The second stage is aimed at improving the noisy output of the matching process, by means of local optimization. A quasi-physical system, inspired by springs and magnets, corrects outliers and estimates better results for uncertain pixel groups. Assuming that most pixels will be correctly matched, an estimation is made on which pixels are more important than others, so less certain pixels and outliers can be corrected to a more probable depth level.

Chapter 2

Methods

This section describes the methods and resources used in the implementation and exploration of the proposed model. Images in the real world should be properly aligned, have the same color settings, and have more properties like these that are related to the world's continuity. To abstract from these difficulties in the model, a virtual world is more appropriate and controllable.

A popular modification to Valve's Source engine [Val04], called Garry's Mod [New06], is a suitable working environment was found in a. Valve offers a extensive set of tools, many of which provided through the bundled Source SDK, to allow the community to develop custom levels, textures, models and modifications. Also, the Source engine provides great flexibility when it comes to in-game manipulation through the developer's console interface.

Garry's Mod extends the user's possibilities for manipulating the in-game world, and adds support for the Lua scripting language [IDFF96]. To simulate binocular vision in this virtual world an adaptation to a Lua script [Sto09] is used to render two viewports simultaneously to the computer screen. Each viewport represents the world as seen by one eye that is slightly shifted horizontally from the other. It is then possible to store the stereo pair as an image file that can be fed into the stereo processing implementation.

The proposed two-stage model is implemented in Java, using the Eclipse IDE. The left-eye view (which is actually the right half of the input stereo pair) is arbitrarily used as a basis for the depth recognition process. As such, the program seeks the depth information for each pixel in the left image by comparing them to pixels from the right image. This process could, in theory, be done by comparing the right image to the left as well, in order to improve the resulting reconstruction. This may be a suitable subject for later studies.

An elaboration on the used algorithms is provided in the following sections. The specifics on the distinctive stages in the model are discusses separately in *2.2* and *2.3* respectively. Reconstruction of the viewed scene in the world is not an integral part of the proposed model in this thesis. Therefore, only a superficial approach to this problem is demonstrated in the separate *Appendix C*.

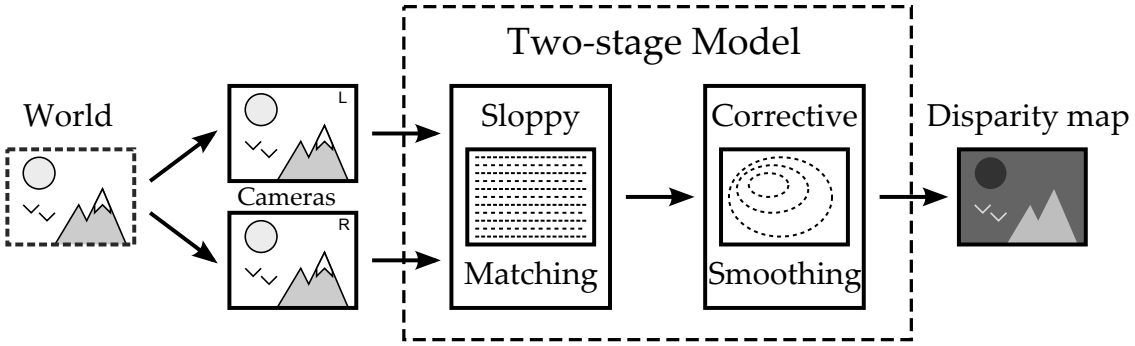


Figure 2.1: This illustration is a simplified workflow diagram of the two-stage model. Two cameras capture the world, the Sloppy Matching stage calculates a rough disparity map, which is then improved by Corrective Smoothing. The output represents the depth information that is derived from the two input images.

2.1 Preprocessing

The image file containing the stereo pair is loaded into the program and stored in such a way, that the individual pixels and scanlines from both views are easily accessible. The left view, which is used as the basis to the reconstruction, will be converted into a texture later on in the reconstruction phase.

The original color values for each pixel in the images are stored as a vector in a color space that is based on human vision [ST02]. As it can be rather complex — and somewhat beyond the scope of this project — to figure out all necessary details concerning an appropriate color model for the operations that are performed on the images, some abstractions are in order. Thus, the images are first converted to grayscale, so that each pixel is assigned a single intensity value. This comes at the cost of losing some information about all pixels, e.g., contrast information is altered, but it greatly simplifies the model.

For each pixel p at position (x, y), the intensity value intensity(p) between 0 and 255 is calculated by a weighted summation of the original color values (also between 0 and 255). The weights are according to the NTSC video standard, originally intended to allow color images to be rendered appropriately on black-and-white television sets [BE04].

$$\text{intensity}(p) = 0.30 \cdot \text{red}(p) + 0.59 \cdot \text{green}(p) + 0.11 \cdot \text{blue}(p) \tag{2.1}$$

2.2 Sloppy Matching

Because the exact projections from the world are lost in the grid of pixels, another way has to be found to determine the disparity for each point in the image. Pixels that correspond to the same points in the world are more likely to have roughly the same intensity. Therefore the simplest way to estimate where a pixel would "fit" in the other image is to calculate which pixel from the other image is most similar. These similarities can, for example, be measured by color values or intensity, by relative position, or by neighborhood.

In the first stage of the model we employ a common approach to finding a small, more or less distorted, part of an image, in a larger image, called template matching. The smaller image, called the kernel, is compared to each possible subimage from the larger image, called the template, to see where it fits best.

There are several algorithms that can be used to calculate a measure of match or mismatch between the kernel and a part of the template. The intensity difference measure that is used here is the sum of the absolute valued differences, or SAVD (*Equation 2.2*). It has very much in common with the sum of squared differences (SSD, *Equation 2.3*), or squared error, but its measure of match is linear rather than quadratic. Both these approaches are equivalent to the root mean square distance (RMS, *Equation 2.4*), but they require less computation.

In the equations related to digital imagery and its mathematical principles, the following definitions will be used for notational convenience. The letters l and r denote the left and right eye image respectively, while k and t denote a kernel and template. Width w and height h limit the array of pixels in an image, kernel or (part of a) template. A single pixel's intensity, for example k_{yx}, has coordinates $0 \leq x < k_w$ and $0 \leq y < k_h$.

$$\text{SAVD}(k,t) = \sum_{y=0}^{h} \sum_{x=0}^{w} |k_{yx} - t_{yx}| \tag{2.2}$$

$$\text{SSD}(k,t) = \sum_{y=0}^{h} \sum_{x=0}^{w} (k_{yx} - t_{yx})^2 \tag{2.3}$$

$$\text{RMS}(k,t) = \sqrt{\frac{1}{h \cdot w} \sum_{y=0}^{h} \sum_{x=0}^{w} (k_{yx} - t_{yx})^2} \tag{2.4}$$

To estimate the horizontal disparity for a single pixel in one image, a kernel of neighboring pixels is matched to the other image. The relative displacement to the position of the smallest mismatch score is the disparity value for that pixel.

2.2.1 Windows

Whereas many conventional algorithms for intensity difference measures in digital image processing rely on comparing pixels in two (spatial) dimensions, i.e. comparing blocks of pixels rather than lines or single pixels, this may not be the most optimal approach in terms of computational complexity for this model.

Because the viewpoints in this model are only slightly shifted along the horizontal axis, the vertical discrepancies in the images (caused by objects being closer to one viewpoint than the other) are minimal. This means that pixels from one image are bound to correspond to pixels that are on the same row in the other image. Therefore, an algorithm that only compares pixels from rows of the same vertical offset will deliver similar results,

whilst providing a significant performance boost. This results in an adapted version of SAVD, to compare one dimensional images, i.e. rows of pixels.

$$\text{SAVD}(k,t) = \sum_{x=0}^{w} |k_x - t_x| \tag{2.5}$$

This type of kernel will from now on be referred to as a window. This method of comparing one dimensional windows is much faster with respect to variants that compare two-dimensional kernels. To elaborate on the workings of template matching using SAVD more intuitively, an example is given in *Figure 2.2*. The disparity value for one pixel from the rows in *Equation 2.6* is calculated.

$$l = \{2, 1, 4, 3, 8, 5, 2, 4\} \tag{2.6a}$$
$$r = \{2, 1, 3, 8, 5, 4, 2, 4\} \tag{2.6b}$$

The matching process for the window $k = \{3, 8, 5\}$ from the left row l to the template r, consisting of the right row, goes like this:

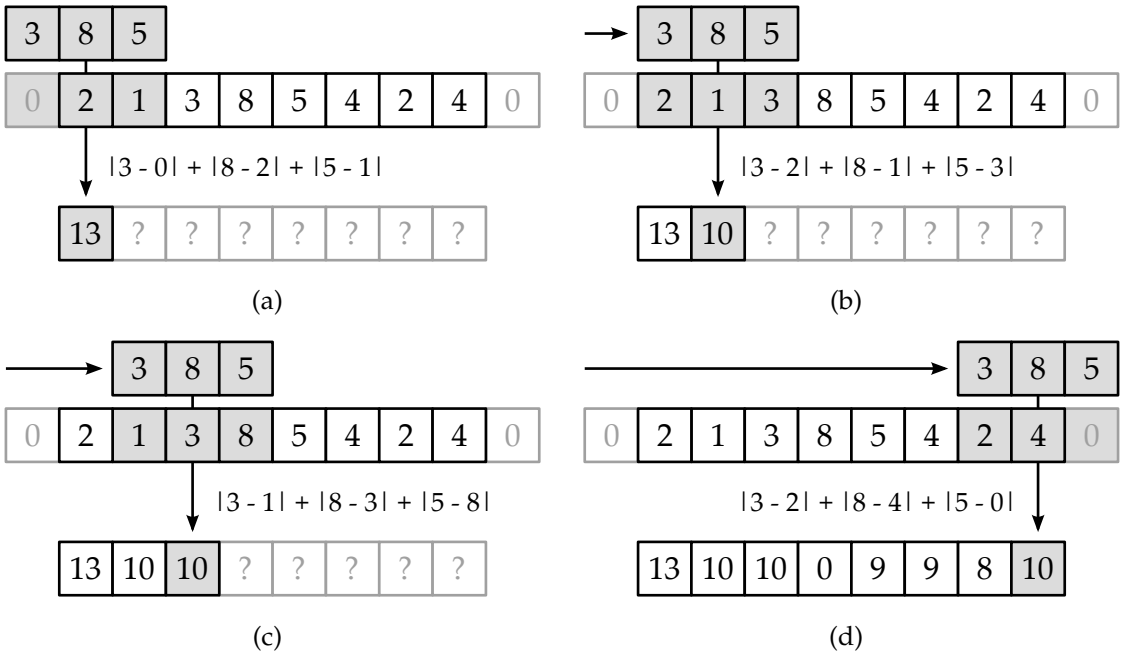


Figure 2.2: These figures illustrate the steps that are taken to match a window $k = \{3, 8, 5\}$ to a template $r = \{2, 1, 3, 8, 5, 4, 2, 4\}$ by calculating the Sum of Absolute Valued Differences (SAVD) for every possible location. This is visualized by the window sliding from left to right, as depicted in (a) through (c), producing SAVDs as it goes. In (d) all SAVDs have been calculated, with the lowest of them all (0) indicating the center of the best matching location in r.

It may be clear from the calculations above, that the window matches best to the fourth position (index 3) in the template, because the difference is minimal there. The disparity

for the pixel at the fifth position in the left row is -1, because that is the difference between its location and the position at which the kernel matches best. Repeating this process for all other kernels gives similar disparity values for the other pixels.

The zeroes that surround the template are there to ensure that SAVDs can be calculated when part of the window is "sticking out" of one side of the template. The number of zeroes required on either side is half the width of the window, because the window's center is never slided beyond the template's limits. It is also possible to fake a continuation of the image the template was taken from, by using a mirrored version of the template instead of zeroes. Such alternative methods may be a suitable subject for further research.

When the disparities for each pixel in an image are calculated, the resulting map can be displayed as a grayscale image with darker pixels for small disparities and brighter pixels for large disparities. As the disparities are directly related to each pixel's relative distance, this image can be treated as a crude depth map.

Calibration and correspondence problems are of greater impact when using one dimensional windows instead of two-dimensional kernels, as only a limited number of the closest neighboring pixels are taken into account. Vertical distortions in the depth map will occur, as the vertical relations of pixel intensities between rows are not taken into account. This will result in a somewhat jagged appearance for edges that are not horizontal.

The basic idea of the sloppy matching stage is to get a quick estimation of the disparity map. This is a matrix that contains the disparity values for each pair of corresponding pixels in the left and right image. To allow a pixel's neighbors to help in finding the correct match, while also keeping the process light and fast, the window size, which in this case is the number of pixels on a line that is matched to a row of pixels in the other image, must not be too large, nor too small. A window size of 5, which means 2 neighboring pixels on both sides of the targeted pixel, is shown to deliver the good results on both fronts, so this value is set for all trials.

2.2.2 Geometry constraints

Because the model is oriented on one image, while using the other image to calculate disparities, there are certain constraints on these disparities that follow from the camera alignment.

Both cameras are modeled to be perfectly parallel to each other, i.e. not focusing on any particular point in the world, but staring straight ahead. Therefore any projected point in one eye, that is visible to both eyes, can only be shifted in one direction relative to its projection in the other eye. This means that not every window for every pixel in a row has to be calculated, but all pixels beyond the current pixel can be pruned from the search, as they can never result in a correct match. This leads to better performance and higher accuracy.

To further enhance performance and limit false positives one can decide to define a maximum disparity value. Pixels that have a larger disparity will be pruned from the search. As larger disparities indicate points that are closer to the cameras, setting this

parameter too low will result in erroneous results for pixels that are too close.

On the assumption that objects in the world are not very likely to "switch" positions in the images because of parallax effects, pixels are very likely to be corresponding to the best match that is also closest. Closer matches are therefore favored above equivalent matches of greater disparity. Unfortunately, occluded pixels in both images, or thin objects switching positions due to parallax, can produce false positives.

2.2.3 Interpolation

Template matching is a method of matching pixels to pixels, and as such, it will return integer values for the positions at which the matching score is best. Effectively, this means that disparities can only be estimated on a per-pixel basis, without sub-pixel accuracy.

In the depth map this phenomenon manifests itself as areas of pixels that have more or less the same disparity values, even though they should be gradients according to the fact that they may correspond to flat surfaces at an angle in the world.

Because of the continuous nature of points in the world, it makes sense to try and simulate this continuity in the images. By interpolating the intensity values on a per-row basis, the pixels in the window can be compared to a template with values that are shifted by less than a pixel. An implementation for fast cubic spline interpolation is used [Fla10]. The program enables this form of sub-pixel accuracy by an interpolation factor that determines how small the steps are, that the window shifts each iteration.

An interpolation factor of 1, corresponds to a step size of 1, which means no interpolation takes place. Higher values correspond to smaller step sizes, i.e. 2 corresponds to steps of $\frac{1}{2}$, 3 means $\frac{1}{3}$, and so on. Interpolation factors higher than 4 no longer seem to be cost-effective, even though the computation time increases only linearly with each step.

The ground truths that are used for evaluation are grayscale coded for disparities. A value of 0 is reserved for unknown pixels, then each increased value corresponds to a disparity change of 0.25 pixels. So with a maximum value of 255, the disparities must lie between 0.25 and 63.75. This ground truth accuracy goes hand in hand with an interpolation factor of 4 that simulates $\frac{1}{4}$ sub-pixel accuracy. This interpolation factor is used on all trials.

2.3 Corrective smoothing

The first stage of the model provides a computationally fast way to find a solution to the correspondence problem. However, information about relations between rows is lost in leaving vertical neighbors out of the calculations. This produces jagged edges and noise in the resulting disparity map. Also, pixels that are occluded in one of both images can cause pixels to mismatch. Thus, regrettably, a considerable number of pixels will fail to match correctly to the other image. These mismatches are visible in the depth map as dark or bright noise.

To compensate for mismatches that remain from the first stage of the model, the second stage aims to eliminate these outliers by means of a form of dynamic constraint satisfaction for local optimization. This method employs a set of simulated, quasi-physical elements that work together to enforce certain constraints over a number of cycles. The following constraints should be taken into account, however soft they may be:

- Outliers are bad. They are probably noise and should go with the flow with their neighbors, unless they are part of said occlusions or very small objects.
- Pixels are unlikely to have disparity values that differ more than 1 from their neighbors. This usually only happens when pixels lie close to a depth discontinuity, or if they are occluded or invisible in the other image.

Pixels that are part of densely textured areas with many different intensity values relatively close to each other, are very likely to match correctly as their neighbors form a distinctive pattern that is easily recognized in the template matching. The same goes for pixels that are part of a sharp edge in the image, often coinciding with a depth discontinuity, in which the two (or more) areas of varying intensities are recognized with ease.

These pixels are very important to the matching process, because they give more information about the correct matches than their less certain neighbors do. A very simple algorithm, often used as a method of edge detection, takes the horizontal derivative of the original grayscale image. This derivative is, in effect, a priority map that shows to what extent all pixels have the desired properties. Pixels that come from textured areas, as well as pixels that lie on horizontal intensity edges, which have probably been matched correctly, have higher derived values. This priority value is given a minimum value of 0.01, so that no pixels will float about aimlessly, unless this behavior is wanted because they are pulled out of range of their magnet (because they are probably erroneous).

In *Sections 2.3.1* and *2.3.2* a method is described that uses this priority map to pull their neighbors and outliers to more suitable locations.

2.3.1 Springs

From here on, pixels shall be seen as small blocks that are placed next to each other as they would be in an image. Each pixel has a spring coming right from its center (in this model springs can pass right through solid objects) attached to its left and right neighbor, as well as to its neighbors above and below it, but not diagonally (as illustrated in *Figure 2.3*). The outermost pixels lie at half a pixel's distance from the walls of the box to which they are connected with a horizontal or vertical spring. So there is an array of interconnected blocks forming an elastic blanket. The springs are in a relaxed state when they are of one pixel's length, when stretched they pull back, when pushed they push back.

Now suppose one pixel is pushed to the right. The springs will pull its left neighbors along to the right. At the same time the spring on the right pushes its right neighbors closer to each other. Note that this model also allows pixels to pass through each other.

Pixels from the next and previous rows are pulled along in a similar fashion, but all pixels are only allowed to move horizontally: all vertical forces are discarded. This elastic behavior lets the pixels smoothen out their relative positions as they are individually pushed sideways using varying amounts of force.

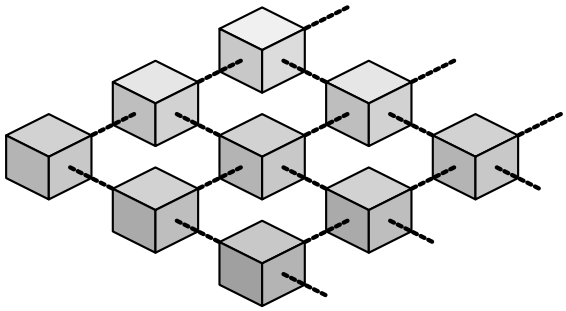


Figure 2.3: A section of the grid of interconnected "pixel boxes" (not to scale). Each pixel is connected to its direct neighbors with a spring.

The net force from the springs, which is unlike physical force in that it only has a horizontal component, on a single pixel, is equal to the sum of the forces of all four springs multiplied by a constant. This constant is further explained in *2.3.3*. Each spring's force is equal to the difference between that springs current length and its length in a relaxed state. If a force is directed to the left, it is regarded as a negative force, so that the net force can be zero if all springs neutralize each other.

The over-all point of this approach with springs and forces is to propagate the coherence of groups of pixels. The differences in disparity between pixels that come from the same object in the scene, are likely to be relatively small. Pixels from different objects, on the other hand, are more likely to be from different depth levels. If a pixel is pulled to one side, it means that the disparity value for that pixel is large, i.e. it is at a distinct distance from the cameras. The net force by the four neighboring springs n on pixel p is defined using the constant c as:

$$F_S(p) = c \cdot \sum_{s=0}^{n} \text{force}(p_s, p) \tag{2.7}$$

2.3.2 Magnets

When only the springs are present, the blanket of pixels will even itself out smoothly, so that each pixel is positioned like in the original image and all springs are at rest. Now imagine an array of magnets of the same dimensions as the pixels, lying at distance 1 under the bottom of the box. Each pixel has one magnet right beneath it. These magnets can only attract one pixel, the one they lie beneath, and they do not affect any other pixels. Also, as with the pixels before, the magnets are allowed to pass through each other, but can only be moved horizontally.

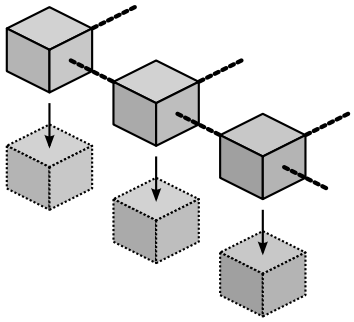


Figure 2.4: Part of a row of pixels with each pixel on top of its own magnet, positioned as if there were no changes in disparity in the image.

So far, the springs and magnets are positioned like in *Figure 2.4*. Now each magnet is shifted horizontally by a distance that is equal to the disparity value of its corresponding pixel. As a result, it pulls its pixel in that direction. If each pixel were to shift right on top of its magnet, the resulting displacements are equivalent to the original disparities. The pixel's displacements can thus be stored as a depth map.

Each pixel can be pushed or pulled by their magnets with a maximum amount of force that is determined by a baseline and a priority value, as described in *2.3*, and it decreases cubically with the distance to its magnet. The baseline value b, set to 0.1, effectively functions as a minimum priority value for all pixels, which prevents magnets from having no effect at all. The magnet m's force for pixel p is defined as:

$$F_M(p, m) = \text{hdist}(p, m) \cdot \frac{\text{priority}(p) \cdot (1 - b) + b}{\text{dist}(p, m)^3} \tag{2.8}$$

The numerator in this fraction represents the importance of the current pixel-magnet pair. The priority is scaled by $1 - b$, allowing the maximum value of $1 = 1 \cdot (1 - b) + b$ and a minimum of $b = 0 \cdot (1 - b) + b$.

The denominator consists of $\text{dist}(p, m)^3$, which represents the cubic decrease of magnetic force over distance, taking into account that the pixels are at distance 1 above the magnets. The third power causes a zero-crossing of this function in the origin — at the magnet's position. Therefore, the horizontal component of the pixel-magnet distance, indicated by $\text{hdist}(p, m)$, is a negative value. The 'vertical' distance between the pixels and magnets ensures no unlimited forces can occur, because no distance would allow divisions by zero.

In this set up, the pixels that are more likely to have matched correctly, are pulled towards their magnets more strongly. At the same time, the less important and error prone pixels are pulled towards their magnets with less force. This allows the prioritized pixels to pull their neighbors along to their correct positions using the springs.

2.3.3 Relaxation

This second stage of the model as described above is rather continuous by nature. To employ its corrective capabilities it must be implemented in a discrete manner. In order to do this, the positions of the pixels are updated in each of a number of cycles. The net forces of springs and magnets are summed up and the result is the distance that pixel will travel that cycle. The net force from the springs is, actually, weighted by a constant factor of 0.1. While this slows down the relaxation process, it prevents the model from exploding when pixels keep overshooting their optimal positions further and further. This happens when outlying pixels are being pulled back too rigorously by their neighbors, causing ever greater ripples that throw the simulation into a chaotic state.

All pixels are shifted to their new positions during one cycle. In the next cycle all forces must be computed all over again as the effects of the magnets are non-linear. As more and more pixels from the same area are drawn in a certain direction, their neighbors may "jump" to their correct positions when they are out of reach of their erroneously positioned or unimportant magnets.

When the magnets and springs, relaxed or not, are starting to reach a stable state, the system reaches its best solution, for the given parameter settings. As soon as the largest displacement between cycles is lower than the tolerance parameter, which is set to 0.01 as a compromise in the trade-off between quality and computing time, the second stage of the model is finished and the improved depth map is stored.

2.4 Datasets and quality measures

Although the two-stage model is designed with the aforementioned in-game world in mind, some method of control has to be implemented to ensure the scientific relevance of the results. The Middlebury stereo datasets [SS02, SS03], a widely endorsed set of stereo image pairs with ground truths, provides this method of control, along with statistics on the performance results of many different algorithms that are used throughout the literature.

The stereo images from these datasets are created with a single digital camera that can be slided horizontally along a metal frame, that also supports a beamer. They developed "a method for acquiring high-complexity stereo image pairs with pixel-accurate correspondence information using structured light" [SS03]. The beamer projects series of structured light images onto a scene, which is then photographed from different horizontal positions. Many pixel correspondences can be derived by decoding the projected patterns.

To compensate for imaging imperfections and include occluded pixels a second phase is done in which the illumination source is shifted horizontally. The disparity values created in this phase also provides a control mechanism for the first phase. By combining imagery from both these phases (and correcting the result by hand), a very accurate and nearly complete disparity map is obtained. The disparity levels for certain pixels cannot

be know using this method, so these should be omitted when testing the results from normal stereo algorithms.

There are many different (computational) models for stereoscopy, and each has a distinct set of strengths and weaknesses. Performance on image regions that are characteristically handled in different ways by the various algorithms, should therefore be evaluated independently from performance on the "whole" image. Regions near depth discontinuities and such require different strategies in terms of correspondence and occlusion.

So besides the images from in-game screenshots, that may deliver visually pleasing results that cannot as readily be evaluated for correctness, four stereo pairs from the Middlebury stereo datasets are used, that come with extensive quality measures. These images go by the identifiers "tsukuba", "venus", "teddy", and "cones", with each name loosely referring to objects in the scene or the picture's origin ("tsukuba" is courtesy of the University of Tsukuba). The images from in-game screenshots that will be discussed are named "car", "interior", and "tree".

Chapter 3

Results

The implementation of the model is based on and inspired by stereo screenshots captured in a game that is powered by Valve's Source engine, because this provides a great deal of control over the scene that is captured, during the development process. To show that the model has any scientific significance, however, a measure of control is necessary that is, unfortunately, not trivial to attained with in-game screenshots. Therefore, a number of stereo pairs from the 2001 [SS02] and 2003 [SS03] Middlebury Stereo Datasets are also fed into the model. They are labeled "tsukuba", "venus", "teddy", and "cones". These images come with ground truths for their disparity maps, so that different algorithms may be quantitatively compared to each other.

Figure 3.1: The "teddy" stereo pair, as for cross-eyed viewing. This means that the left eye's image is actually on the right, and vice versa, so that practised viewers may cross their eyes and trick their minds into merging the separate views into a consistent three-dimensional image.

Every one of the stereo pairs is fed into the program as a single image file that contains the views from both "eyes", stitched together for portability, like in *Figure 3.1*. The program then separates the two views and analyzes one in relation to the other. For each

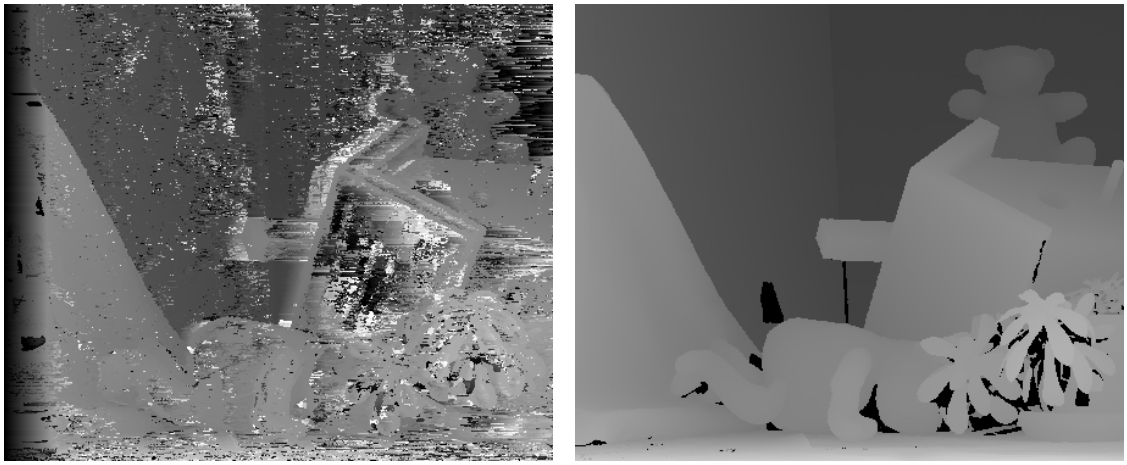
of the model's two stages, the program will output a separate result. These two results are discussed in the following sections for the "teddy" stereo pair as a guiding example. More complete resulting data can be found in *Appendix B*.

Also, the result of both stages is compared in terms of percentages of bad pixels. These percentages are determined for a number of "badness" thresholds, so they can be plotted to compare the performance of the model's stages to each other and to the performance of some other algorithms. These data are available for the Middlebury images through their website [Sch07].

3.1 Sloppy matching

The first stage performs a fast, but sloppy, form of template matching, using only lines for windows instead of boxes, whilst reducing the correspondence problem to a one dimensional search problem.

The resulting disparity map (*Figure 3.2*) is mostly as expected: many pixels in textured areas and areas in border regions are matched correctly, about 17 percent of all known pixels in this image is even exactly spot-on (see *Figure 3.5*), which is about average for the four stereo pairs that were evaluated using their ground truths. Performance in occluded regions is slightly lower than the overall result: when occluded areas are not counted the average of pixels that are exactly right becomes more than 18 percent. Exact matches near depth discontinuities average out at about 17 percent as well. It seems that the specific image region makes only very little difference, in terms of exact matches.



(a) Sloppy matching result. (b) Ground truth.

Figure 3.2: Sloppy matching result for "teddy" and its corresponding ground truth. Note the rather large erroneously matched regions on the birdhouse's roof and to the right of the teddy bear.

Most pixels are not matched exactly correct, but that does not necessarily mean they are all bad. When taking a threshold error score, it becomes clear that there are quite a lot of "near misses". Scharstein states the following on the Middlebury stereo website:

> "The old, fixed, error threshold of 1.0 has been problematic since it did not distinguish between integer-based methods and those that compute sub-pixel estimates. Even worse, on the Tsukuba images (which only has integer ground-truth disparities) it favored rounding disparities to integers since a disparity error of, say, 1.4 would be considered a "bad pixel", but not when rounded to 1.0. The new variable error threshold allows comparison of sub-pixel performance (when 0.75 or 0.5 is selected), and discourages rounding (if 0.5 or 1.5 is selected). The default error threshold that determines the "official" rank of each algorithm in the table is still 1.0" [Sch07].

Figure 3.5 demonstrates this sub-pixel effect for the "teddy" images using different error thresholds. When culling the disparity errors to the "official" threshold of 1.0 (t = 1.0), instead of the aforementioned percentages for a threshold of 0.0, the error scores and averages for the different image regions are as follows:

t = 1.0	non-occ	all	disc
Tsukuba	28.87	30.54	28.66
Venus	35.34	36.42	35.58
Teddy	37.95	44.28	43.00
Cones	41.94	48.29	44.24
Average	36.02	39.88	37.87

Table 3.1: The percentages of bad pixels for the sloppy matching stage with a threshold of 1.0, lower is better. All four images are evaluated on non-occluded regions, all (including half-occluded) regions, and regions near depth discontinuities.

These results don't seem to be very promising, but that is actually the point in this two-staged model! The first stage only has to perform as good as to allow the second stage to correct the erroneous matches, while keeping computation time really low. *Table 3.1* and *Figure 3.5* show that the matching process performs best in non-occluded regions, while performance on regions near depth discontinuities is slightly less. However, the sloppy matching stage does seem to provide a usable, although noisy, disparity map.

3.2 Corrective smoothing

The second stage in the model aims to enforce continuity constraints while favoring pixels that are probably matched correctly. All pixels are laid out in a grid, interconnected by springs, and each pixel is individually pulled to the position it was initially matched to, in the first stage, by its own magnet.

Pixels that are matched too far from their neighbors, and have a low priority value (see *Figure 3.3*), are pulled out of range of their magnet to a more probable point in between their neighbors that makes the disparity map more smooth. This means less important

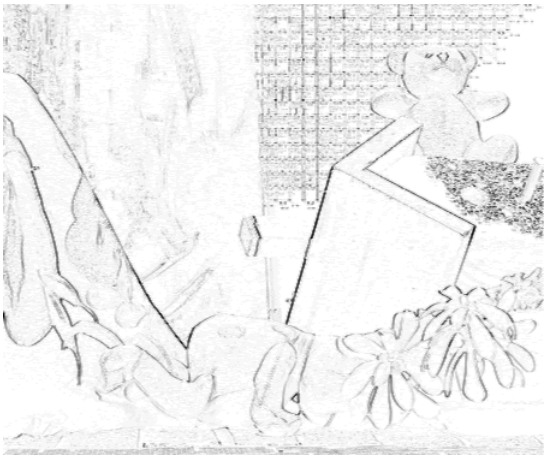


Figure 3.3: The priority map for the "teddy" stereo pair, inverted for better visibility. Darker values denote more important pixels, and are determined by the intensity difference between pixels and their horizontal neighbors.

pixels, which are more likely to match badly, are more likely to be corrected than more important pixels, which are more likely to match badly and help in correcting the others.

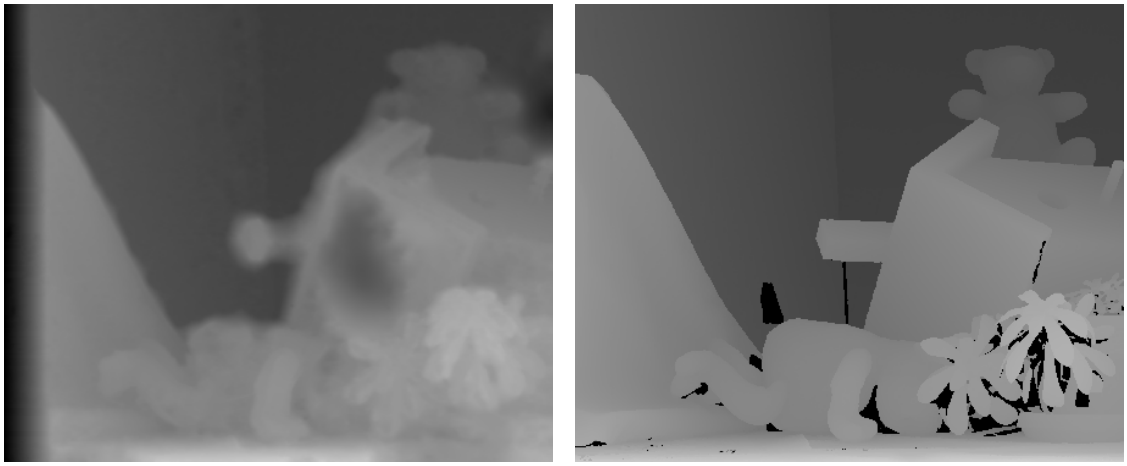
Unfortunately, since the springs and magnets cause the disparity map to no longer be rounded to a disparity value accuracy of 0.25, there are no longer any exact matches with the ground truth either. *Figure 3.5* illustrates this by each "smoothing" line starting at a 100% error score instead of the 83% of the "matching" lines. This may seem not to bode well for the overall corrective performance of this stage, but let's ignore the exact matches for now and increase the error threshold to 1.0.

The resulting error scores (*Table 3.2*), suddenly are much better than those from just the matching process. Most improvements are in the non-occluded areas, and whereas the areas near depth discontinuities are worse for "tsukuba" and "venus", they are better for "teddy" and "cones". This may be attributed to the former two images being simpler, having many straight lines for depth discontinuities, while these are more erratic in the latter. Occluded regions are not handled very well by the corrective stage, so large, continuous occlusions may cause the decreased performance in "tsukuba" and "venus".

t = 1.0	non-occ	all	disc
Tsukuba	9.52	11.45	29.17
Venus	11.95	13.48	38.29
Teddy	20.15	28.40	35.40
Cones	19.09	28.02	34.10
Average	15.18	20.34	34.24

Table 3.2: The percentages of bad pixels for the corrective smoothing stage with a threshold of 1.0, lower is better. All four images are evaluated on non-occluded regions, all (including half-occluded) regions, and regions near depth discontinuities.

With this higher error threshold, it becomes clear that there still may be something



(a) Corrective smoothing result. (b) Ground truth.

Figure 3.4: Corrective smoothing result for "teddy" and its corresponding ground truth. Note, relative to the sloppy matching result in *Figure 3.2*, the great improvement to the birdhouse roof and overall error reduction. For a line-up of all "teddy" results, see *Appendix B.4*.

useful to this corrective stage. *Figure 3.4* is visually more pleasing than *Figure 3.2* in that it seems much less noisy. Rather large erroneously matched areas — such as the birdhouse roof — tend to blend in better with their correctly matched surroundings. Although the inner regions of such areas may still not be correct in relation to the ground truth, they are much smoother and of one color. The outer regions of these erroneous areas are corrected better as these pixels are pulled along with the correct matches in their surroundings.

Some erroneously matched pixels, that lie at a few pixels distance from any correct (and important) pixels, are thus out of range of the corrective influence of their nearest correct neighbors. This might in some cases be fixed by lowering the relaxation threshold (mentioned in *Section 2.3.3*), but this is accompanied by a large increase in computing time, as things only improve very little over a great number of cycles.

3.3 Comparison

Figure 3.5 not only shows the performance of the matching process alone and in combination with the smoothing stage. It also shows data on one of the best and one of the lesser performing algorithms available from the Middlebury stereo website.

The model that is proposed here seems to perform about as well as LCDM+AdaptWgt [NG10]. However, as the SurfaceStereo [BRK10] algorithm shows, it is possible to do a lot better.

Although expected, it is interesting to see how every one of these algorithms turns out to have the most problems near depth discontinuities, while non-occluded areas prove less difficult.

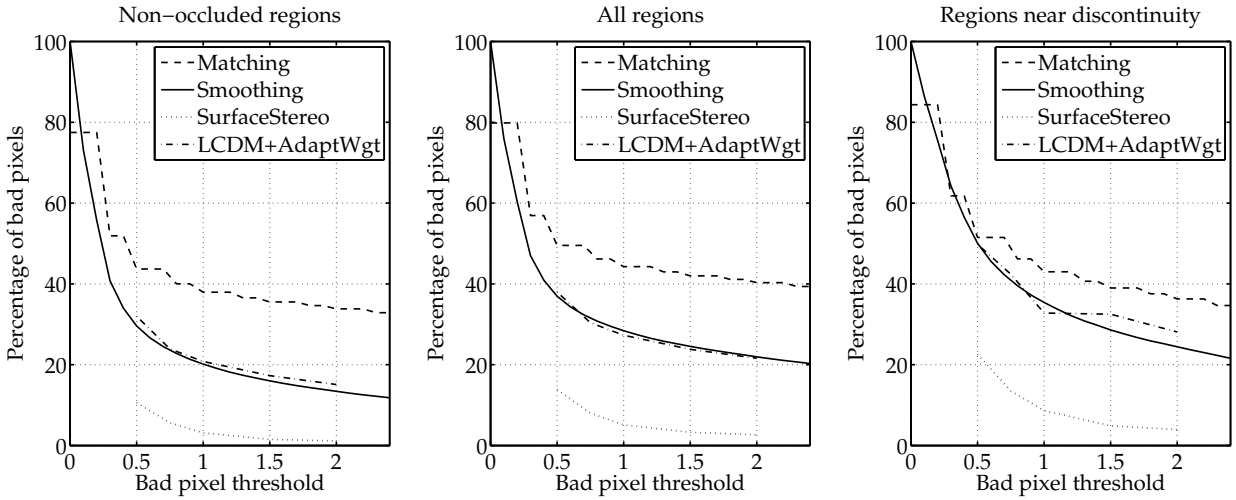


Figure 3.5: Performance in different regions of the "teddy" stereo pair, in relation to two other models. The threshold is increased in steps of 0.1 in terms of disparity errors (bad pixels). Note the decreased performance of the corrective smoothing stage near depth discontinuities, and the increased performance in non-occluded areas.

Chapter 4

Discussion

The main research question in this project is aimed at exploring computational models for depth reconstruction using stereoscopic imagery.

> Is it possible to derive a depth map from two images by analyzing (horizontal) parallax, using a computationally flexible algorithm, so that it can be used for scene reconstruction?

Following the investigation on existing models, a new, two-stage model is proposed, consisting of a "sloppy matching" stage and a "corrective smoothing" stage.

The first subquestion, regarding calibration of the cameras in relation to each other and to the world, is solved by using virtual cameras in the Source engine that are parallel to each other. This allows for the matching process to be treated as a one dimensional search problem. The Middlebury stereo images are pre-rectified, so they can be treated in the same way.

This model's approach mainly focuses on the second subquestion, regarding the correspondence problem. The first stage is very fast, but somewhat inaccurate, while the second stage's purpose is to improve upon the first stage's results by enforcing certain constraints that should hold for the output.

The third subquestion, about reconstruction of the original three-dimensional scene from the calculated disparities, has proven to be too much to study and discuss in-depth in the course of this project. Therefore, a description of associated methods and some of the results are moved to *Appendix C*.

The sloppy matching stage, although rather inaccurate (*Table 3.1*), works as fast as it should. Most pixels are matched correctly, or nearly correctly, while the erroneously matched pixels that occur in non-occluded regions are spread thin.

Occluded pixels are not handled in a neat way, but they are still mostly distinguishable with the naked eye, next to other incorrect matches, so there is room for improvement there. Small occluded regions may appear as gradients, because part of the matching windows is consistently matched to one line of intensity discontinuity, instead of exclusively to the nearer or farther object.

The same holds for large untextured areas, where outer pixels tend to match to that region's border. The inner pixels will either match to infinity — because they seem to match best to their original positions — or to seemingly random positions in range that match slightly better.

The corrective smoothing stage does a good job at correcting sparse, erroneously matched pixels (*Table 3.2*), by pulling them back towards the more reasonable disparities of their neighboring pixels. However, it fails blatantly at correcting depth discontinuities, as it will smear out the disparities to a gradient between the nearer and the farther object — for smaller, as well as for larger occluded areas.

Performance in untextured areas is quite the opposite, as the smoothening effect is very much desirable here. Small monochromatic(-ish) regions will be corrected from the outside in, but in larger regions the innermost pixels are not under such positive influence, because they lie too far from any correctly matched pixel's area of effect. Nonetheless, they do smoothen out along with their more direct neighbors, albeit at a non-desired disparity level.

4.1 Conclusion

The sloppy matching stage performs almost equally well in most regions of the images it is used on in this project. On average, error scores are worst when evaluating "all regions" of the output (that is, excluding unknown disparities and including occluded and half occluded regions), slightly better near depth discontinuities (as only about half of those reveal occluded pixels), and best in non-occluded regions. Still, its fast execution (a matter of seconds, using single-core processing) makes it an attractive way of getting a sketchy disparity map.

The corrective smoothing stage has come out not so much an independent stereo matching algorithm, but a system that tries to emphasize certain features that are defining in a correct disparity map (as described in *Section 2.3*). Smoothness constraints are enforced, that do not only remove noise, but also attempt to correct not-too-large areas that are erroneously matched due to lack of texture or occlusions.

While the first stage of the model on its own is not very good at producing correct disparity maps, especially in comparison to many other algorithms that are out there, the model's performance improves greatly when the second stage is added to the process. And although the overall model's results may not be as good as those of its rivals, the corrective smoothing capabilities are quite promising. Also, the corrective smoothing stage is quite portable, as it is only loosely coupled to the first stage.

4.2 Further research

In its current form, the model has proven effective only to some extent. Therefore, as its performance is not among the best yet, there is some room for improvement. This section

lists a number of principles that may be worth exploring in possible further research.

Occlusions and untextured areas are the image regions that the model has the most difficulties with. Enabling both stages to detect and process these areas separately from "normal" regions may increase performance greatly.

Incorporating color in the matching and smoothing processes, instead of just intensity values. By figuring out a suitable color model, it is possible to use full-color pixels in the stereo matching process, instead of just the compiled intensity values. This should improve results, because the full-color images contain more information about the scene.

Parallel processing. The processes for sloppy matching and corrective smoothing are both specifically designed so that they may be distributed over several threads, as to improve computation time on multi-core computer systems, or GPUs [VN08]. In the matching process, each horizontal line's calculations are completely independent of the other lines, so it is possible to start a separate thread for each line. In a more extreme case, the matching process for a line could even be split into threads for each window separately. After all computations on the first stage are completed, the corrective smoothing stage can also be distributed easily. But whether one splits per pixel or per group of pixels, each iteration is dependent on the last, so each iteration has to finish before the next can start.

Online processing of stereo pairs that are generated by a pair of cameras (as on the moving robot mentioned in the introduction), would be a useful evolution on the current offline model. The quick-and-dirty approach of the first stage might prove more successful in such a situation, because the outputs from the different stereo pairs could be used to form an incremental world model. Each new stereo pair then provides new details about the world that can be used to get a more consistent and correct reconstruction of the world. Should the model prove to be too computationally intensive for an embedded implementation, a solution may be to wirelessly connect the robot to a more powerful computer.

Appendix A

Source code

This appendix holds parts of the source code of the Java implementation that is used in this project. This introductory section contains a short description and explanation, while the actual Java code can be found in *Sections A.1* and *A.2*.

The sloppy matching algorithm selects a window for each pixel in each row from the left image. This window is shifted along an interpolated version of the corresponding row from the right image. At each position the window is shifted to the sum of absolute valued differences is calculated over the pixels in the window with the corresponding interpolated pixels. From all these SAVD's, only the lowest (best) value and its position is stored. If some SAVD is just as good as the current best, but is located closer to the window's origin, it is selected instead. So each pixel from the left image now has a corresponding pixel in the right image which is supposed to be a projection of the same point in the world. All the best positions are stored in a matrix, to be passed on to the next stage. Also, a matrix is created which indicates how important each match is, and how probable it is that it has matched correctly. Unfortunately, this method is not flawless, so many erroneous matches will occur. That's where the next stage comes in.

The corrective smoothing algorithm takes the two matrices "magnet_position" and "magnet_force" from the previous stage. All pixels from the left image are represented in a matrix that is initialized with the same values as "magnet_position". I.e. each pixels starts at its magnet's position. Then an iterating process of updating is started, where each iteration forms a new matrix of pixel positions, based on the outcome of the previous iteration. For each pixel the attractive force from its magnet and the forces from springs, that are attached to its direct neighbors, are calculated. The sum of these forces is then applied as the horizontal displacement of that pixel for the next iteration. This relative horizontal displacement represents the disparity value for that pixel. If relaxation mode is enabled, the maximum change between iterations is stored, and the process ends when this value falls below a predefined threshold. Otherwise a fixed number of cycles can be set.

A.1 Sloppy matching

```
1  // Line-based Template Matching between images using SAVD
2  public double[][] match() {
3    Progress p = new Progress("Matching:", height);
4    int window_size = Stereoscopy.WINDOW_SIZE;
5    double max_disparity = Stereoscopy.MAX_DISPARITY;
6    double step_size = 1.0 / Stereoscopy.INTERPOLATION;
7    CubicSplineFast right; // Interpolated version of the right row
8    double savd; // Sum of absolute valued differences
9    double min_error; // The best (smallest) error score
10   double best_pos; // The best position (with the lowest error)
11   double possible_from; // Limit horizontal matching possibilities
12   // For each row in the left image
13   for (int row = 0; row < height; row++) {
14     p.update(row);
15     right = Interpolation.CubicSplineFast(this.right[row]);
16     // For each column in the left image
17     for (int left_col = 0; left_col < width; left_col++) {
18       min_error = 255 * width;
19       best_pos = -1;
20       possible_from = (left_col - max_disparity > 0) ? left_col
21           - max_disparity : 0;
22       // For each possible location in the right image
23       for (double right_col = possible_from; right_col <= left_col; right_col += step_size) {
24         savd = 0;
25         // For each pixel in the window
26         for (int window_col = -window_size / 2; window_col < window_size / 2 + 1; window_col++)
27           // If we're inside the picture's boundaries
28           if (left_col + window_col >= 0
29               && left_col + window_col < width
30               && right_col + window_col >= 0
31               && right_col + window_col < width)
32             // Calculate the sum of absolute valued differences
33             savd += Math
34                 .abs(left[row][left_col + window_col]
35                     - right.interpolate(right_col
36                         + window_col));
37         // See if this position matches better or closer by
38         if (savd < min_error
39             || (savd == min_error && right_col > best_pos)) {
40           min_error = savd;
41           best_pos = right_col;
42         }
43       }
44       // If no good positions are found, we'll want to know about it
45       if (best_pos == -1)
46         System.err.println("No match found at (" + left_col + ", "
47             + row + ")!!!");
48       // Store the best matching position
49       position[row][left_col] = best_pos;
50       magnet_force[row][left_col] = priority[row][left_col]
51           * (1.0 - Stereoscopy.MIN_PRIORITY)
52           + Stereoscopy.MIN_PRIORITY;
53       magnet_position[row][left_col] = best_pos;
54     }
55   }
56   p.finish();
57   // Convert positions to disparities
58   return result();
59 }
```

A.2 Corrective smoothing

```
1  // Calculate the positions for the next cycle
2  public void update(boolean relax) {
3    // Nothing's changed yet in this cycle
4    max_change = 0;
5    // Prepare to store the next positions
6    double[][] next = new double[height][width];
7    // For every row
8    for (int y = 0; y < height; y++) {
9      // For every column
10     for (int x = 0; x < width; x++) {
11       // Determine positions of self and neighbors and their (linear)
12       // forces on self by taking their horizontal distances
13       double self = position[y][x];
14       double left = (x > 0) ? position[y][x - 1] - self + 1 : 0;
15       double right = (x < width - 1) ? position[y][x + 1] - self - 1
16           : 0;
17       double up = (y > 0) ? position[y - 1][x] - self : 0;
18       double down = (y < height - 1) ? position[y + 1][x] - self : 0;
19
20       // Summarize forces of all springs and scale
21       double springs = Stereoscopy.SPRINGS
22           * (left + right + up + down);
23
24       // Determine the horizontal distance for ``self's'' magnet
25       double hdist = magnet_position[y][x] - self;
26       // And then the Euclidean distance
27       double dist = Math.sqrt(hdist * hdist + 1);
28       // Calculate the magnet's force (decreasing cubically)
29       double magnet = (magnet_force[y][x] / (dist * dist * dist))
30           * hdist;
31
32       // Add all forces to determine the net force on self
33       double force = springs + magnet;
34       // Calculate the next position
35       next[y][x] = position[y][x] + force;
36
37       // If we're in relaxation mode, calculate how much we've changed
38       // in order to determine if we're done yet
39       if (relax) {
40         double change = Math.abs(next[y][x] - position[y][x]);
41         max_change = (change > max_change) ? change : max_change;
42       }
43     }
44   }
45   // Store all new positions for the next cycle
46   position = next;
47 }
```

Appendix B

Comprehensive results

The "Results" *Section 3* contains a part of all resulting data that is representative of the overall outcome of the algorithms. In this appendix, the results of this study are reproduced in more detail.

First a description is given (*Figure B.1(a)*, *B.1(b)*, and *B.1(c)*) of the way in which different aspects of stereo algorithms are tested by evaluating different image regions that represent key elements of the depth structure contained in the images.

Secondly, an overview is given of the disparity maps for the two stages of the model, for the "tsukuba", "venus", "teddy", and "cones" stereo images. Each of these pairs of disparity maps is supplemented with its ground truth and plots that illustrate the model's performance on different image regions. Subsequently, the disparities for the in-game screenshots ("car", "interior", and "tree") are given, unfortunately without ground truth, or quality measures. These measures are not readily available, as the Source engine's depth buffer does not contain sufficient depth information to serve as a control mechanism — it only contains relative depth information for objects that are very close to the camera. It may be possible to develop some ray-tracing algorithm in Lua [IDFF96], but that falls outside of the scope of this project.

In the final part of this appendix, results for two other stereo algorithms are presented for the "tsukuba" image. Using the image regions similar to the ones in *B.1*, the performance of the model's two stages is compared to one of the highest, as well as one of the lowest ranking algorithms.

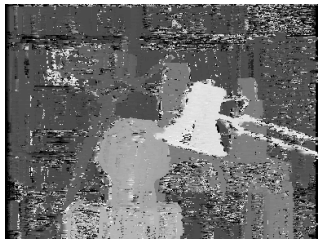
All images used in this thesis project for evaluating the mentioned algorithms in a controlled manner come from the Middlebury stereo datasets [SS02, SS03]. Thanks go out to the people responsible for making these useful images freely available.

B.1 Image regions

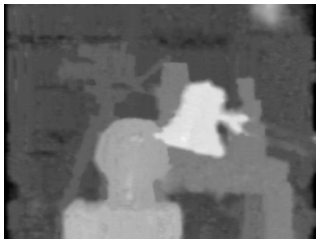
(a) Non-occluded regions. (b) All known regions. (c) Regions near discontinuities.

Figure B.1: An example of different image region subdivisions for the evaluation of stereo algorithms (using the "teddy" image). For each figure holds: errors are only evaluated in the white regions. *Figure B.1(a)* shows non-occluded regions (white) and occluded and unknown regions (black). The second *Figure B.1(b)* shows all known (including half-occluded) regions (white) and unknown regions (black). *Figure B.1(c)*, finally, shows regions near depth discontinuities (white), occluded and unknown regions (black), and other regions (gray).

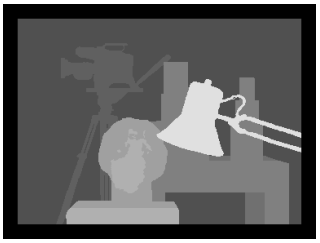
B.2 Tsukuba



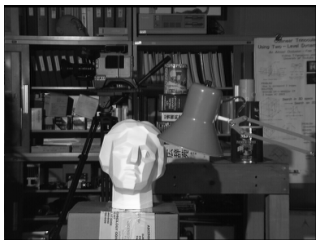
(a) Sloppy matching.



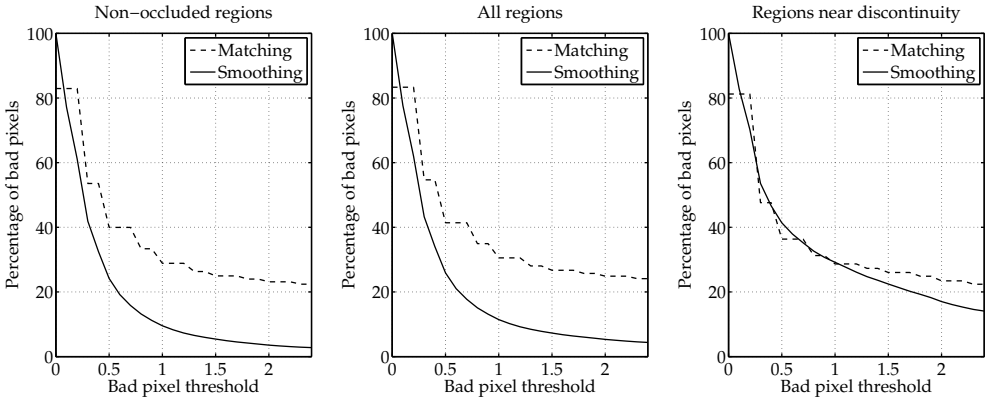
(b) Corrective smoothing.



(c) Ground truth.



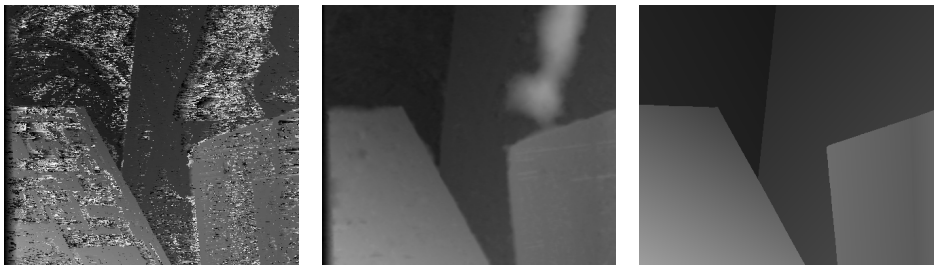
(d) Source image.



(e) Performance on different regions.

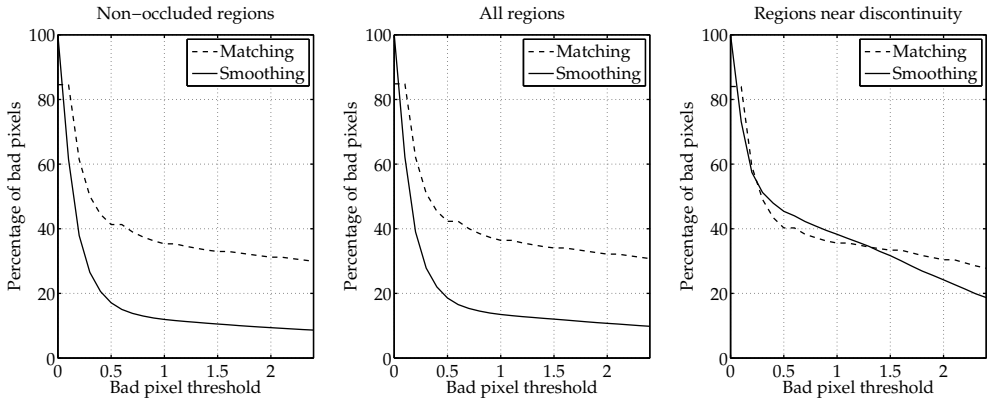
Figure B.2: The ground truth for this stereo pair is only accurate to integer values, so some disparities may actually be better or worse, depending on their sub-pixel accuracy. Note how, in the corrected result *B.2(b)*, the lamp's wire and arm are "smoothened away" by the springs as if they were errors. The table and the box, however, are neatly cleared of noise. Disparity levels are multiplied by 16 to get clearly visible features in grayscale.

B.3 Venus



(a) Sloppy matching. (b) Corrective smoothing. (c) Ground truth.

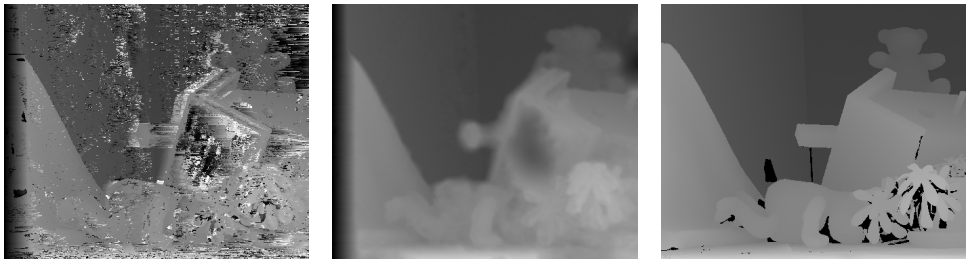
(d) Source image.



(e) Performance on different regions.

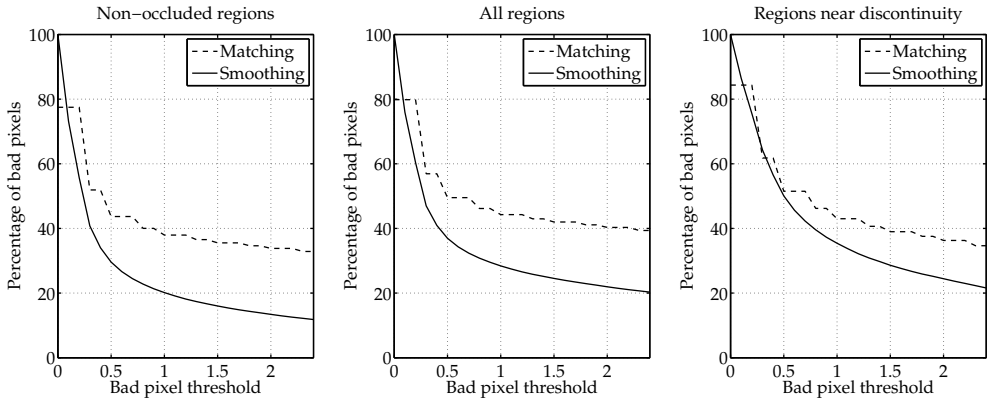
Figure B.3: The dark area above the sports page is a good example of how the smoothing phase tries to incorporate badly matched pixels into a more consistent whole. Whereas the damage to that area is a little too extensive for a complete reconstruction, the two text pages and the background are corrected nicely. Disparity levels are multiplied by 8.

B.4 Teddy



(a) Sloppy matching. (b) Corrective smoothing. (c) Ground truth.

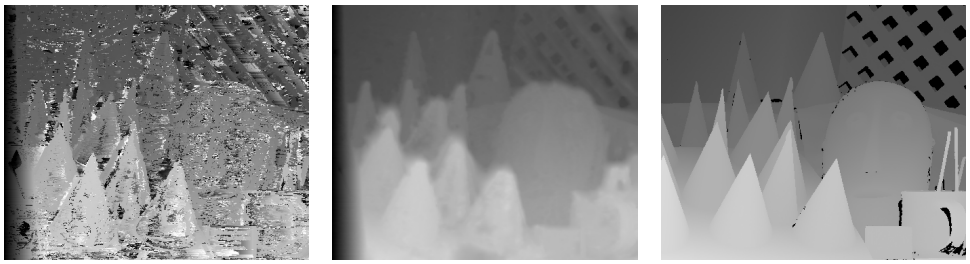
(d) Source image.



(e) Performance on different regions.

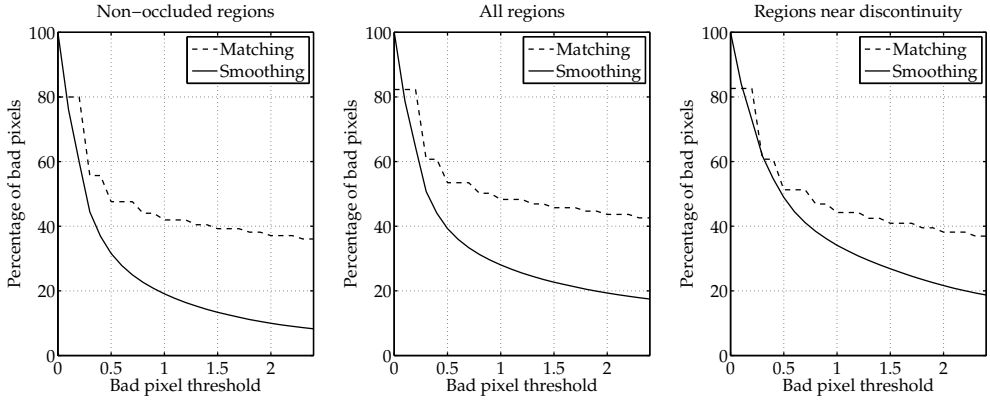
Figure B.4: This stereo pair is discussed in detail in *Chapter 3*. Notable features are the reconstruction attempt for the birdhouse's roof, the occluded area on the far left and to the left of the teddy bear and birdhouse roof. Disparities are multiplied by 4.

B.5 Cones



(a) Sloppy matching. (b) Corrective smoothing. (c) Ground truth.

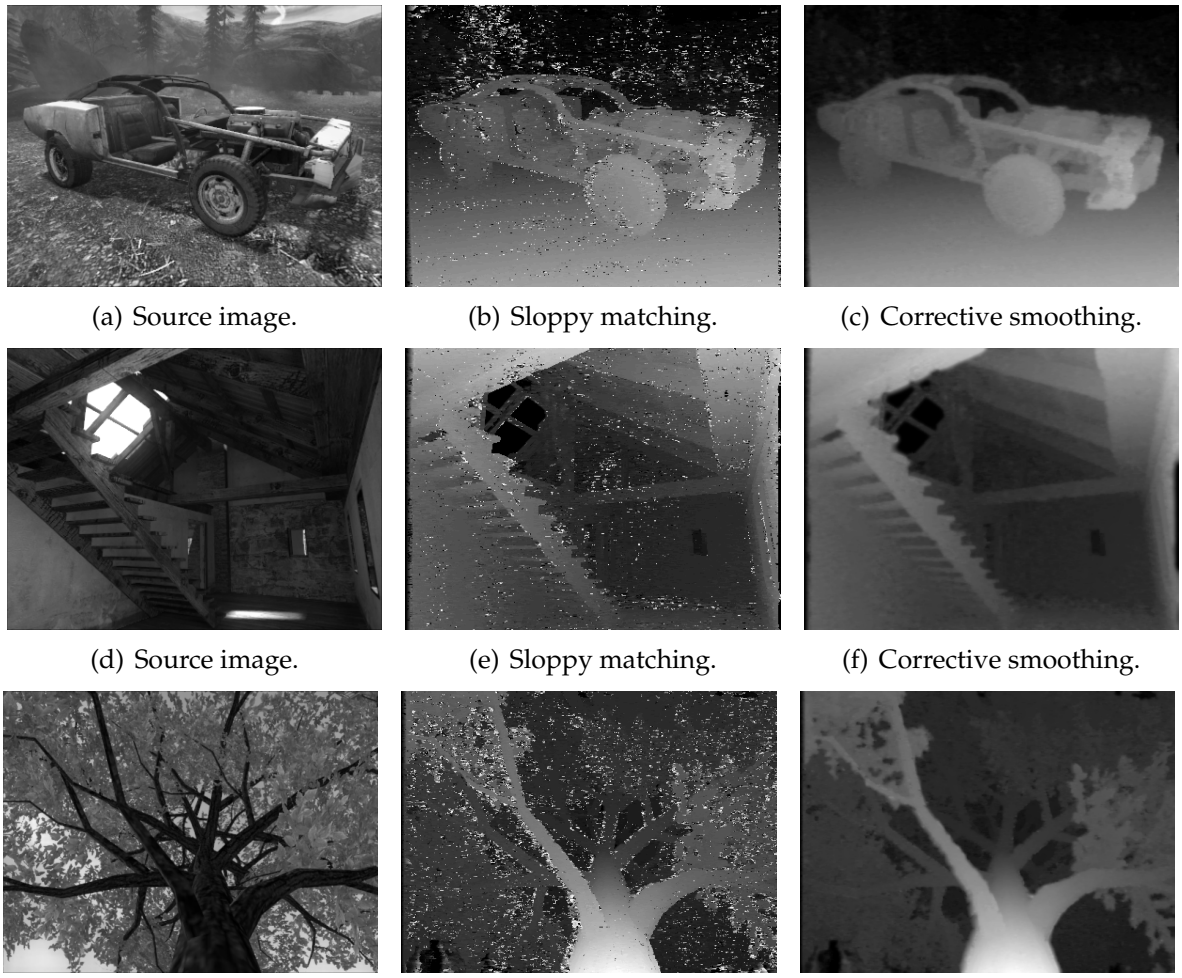
(d) Source image.



(e) Performance on different regions.

Figure B.5: This image demonstrates the corrective power of the model's second stage, while also showing one of its major downsides. Small changes in disparity, near the mask's eyes, nose, and mouth, are visible. But on the other hand the pencils are smeared out and the tips of the cones are no longer sharp. Disparities are multiplied by 4.

B.6 Source engine stereo pairs



(a) Source image. (b) Sloppy matching. (c) Corrective smoothing.

(d) Source image. (e) Sloppy matching. (f) Corrective smoothing.

(g) Source image. (h) Sloppy matching. (i) Corrective smoothing.

Figure B.6: Output for both stages for the Source engine stereo pairs "car", "interior", and "tree". Note how the matching process performs much better on these images, compared to the Middlebury stereo pairs. This is because the disparities in these images are much smaller, which makes it more probable for the windows to match to the correct positions. Also, there is less chaos in the corrective smoothing system of springs and magnets, so it settles down easier. Each disparity map is brightened by an individual constant factor to improve visibility.

Appendix C

Reconstruction

The disparities contained in the depth map are clues to the distance from the points in the world, that are projected into the individual pixels, to one of the cameras' focal point. The relation between disparity values and the actual positions of the corresponding points is defined by the mathematical principles referred to in *C.1*. Because many of the stereo pairs that are used in this project are screenshots from renderings by the Source engine, it makes sense to try and bring the reconstructions back into that engine.

First the disparities that result from the two-staged model are converted to distance measures by means of triangulation. Then, a VTF-file (Valve Texture Format) is created from one of the original images from the stereo pair, to make the reconstruction more viewable. To discriminate between separate objects in the scene, some pixels are rendered transparent, using the texture's properties as defined in a Valve Material Type. This way depth discontinuities can be made to appear "cut out" from farther objects. A three-dimensional representation for the reconstruction is then created in the form of an array of disparity maps, stored in Valve Map Format. For information on VMF and the other specific file formats mentioned above, see [Cor06] and further pages in the Valve Developer Community.

C.1 Triangulation

Through the process of triangulation points in the world can, to some degree, be reconstructed from their corresponding projections. The exact relationship between pixel sizes, different screen resolutions and in-game distances is not sufficiently defined in the game's documentation. Since figuring out these specifics is somewhat beyond the scope of this project, the reconstructed results will be off by a constant factor, i.e. measures will be calculated to screen width percentages.

An important constant in stereo reconstruction is the camera's focal length. Because this property is not clearly defined for the virtual cameras that are used for the original image samples, the optimal viewer-to-screen distance is substituted as defined in the FOV (field of view) documentation [Cor06]:

$$(\text{focal length}) = (\text{viewer distance}) = \frac{(\text{screenwidth})}{2 \cdot \tan(\text{FOV}/2)} \tag{C.1}$$

The reconstructed distance can then be derived using fairly elemental trigonometry. In the end it boils down to the distance being dependent on, besides the horizontal disparity, the inter-eye (inter-camera) distance, and the focal length.

$$(\text{reconstructed distance}) = \frac{(\text{distance between eyes}) \cdot (\text{focal length})}{(\text{disparity})} \tag{C.2}$$

C.2 Displacement mapping

A first approach to representing a reconstruction of the original scene could be in voxels (volumetric picture elements [SD02]). But since the Source engine can only render a total of 8192 of the elements that are required, called brushes, as opposed to the $300 \times 225 = 67500$ of pixels that are necessary for a relatively small image, a different approach is used.

The engine does allow for more pixels to be represented by use of displacement maps. This is a type of deformable surface consisting of interconnected triangle polygons [FL95]. Using this method it is possible to represent $8 \times 8 = 64$ pixels per brush, so only $67500/64 < 1100$ brushes are necessary. Unfortunately, there also seems to be a maximum to the number of displacement surfaces (which is not directly mentioned in the documentation), so images larger than 300×225 may cause problems.

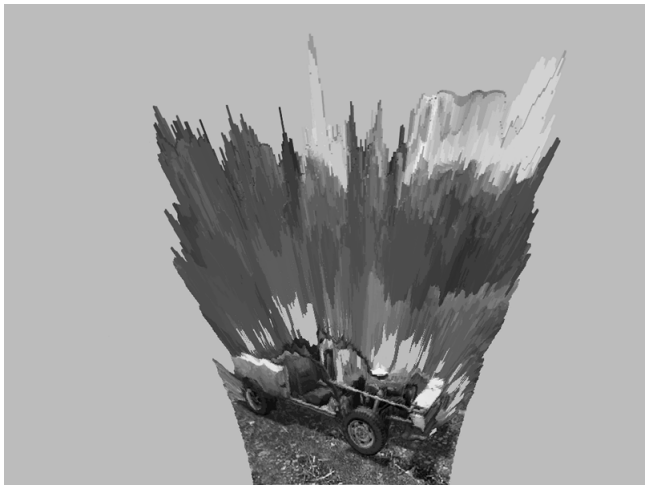
Displacement surfaces consist of $2 \times 16 \times 16$ triangles, where a pair of triangles (forming a square) can represent a pixel. Reconstructed pixels are set one square apart to allow for depth discontinuities as discussed in *C.3*. By manipulating the corner points of the reconstruction squares, the size and relative distance for each pixel can be represented (see *Figure C.1*).

C.3 Texture transparency

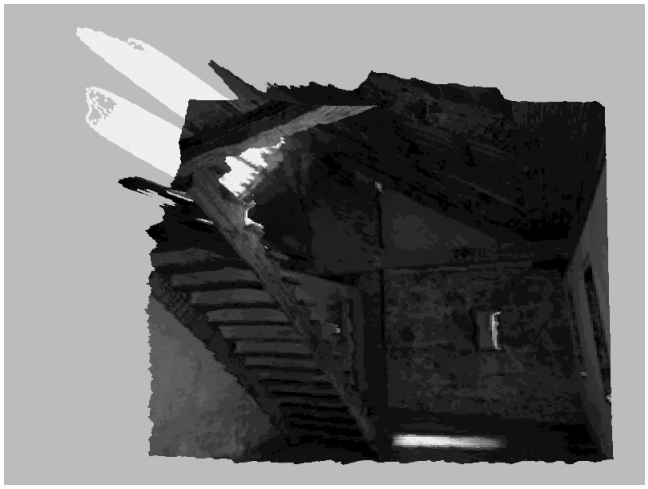
Only points that are visible in the images can be reconstructed. Less visible points, such as point that are occluded, or from separate objects, cannot be reconstructed as accurately and should therefore be treated differently from fully visible points.

Ideally, depth discontinuities are reconstructed by separating the continuous surfaces of different objects from each other. To accomplish this, the polygons connecting the reconstructed pixels are textured opaque (with an interpolated color value) when the change in depth is smaller than some threshold, while they are rendered transparently when they lie on a greater change in depth.

It is probably possible to distinguish depth discontinuities by using more advanced methods, like a combination with segmentation algorithms. Still, the effects of the method described here are sufficient to illustrate the principle.



(a) Reconstruction of the "car" stereo pair. The trees and the sky in the background are indeed much farther away than the car itself and the ground in front of it.



(b) Reconstruction of the "interior" stereo pair. The stairs, and the joist on the first floor they lead to, are nearer than, and occluding, the window. The sky is projected to approach an infinite distance.

Figure C.1: Possible reconstructions for two of the Source engine stereo pairs. To prevent the rendering engine from causing overflows, the images are resampled to a lower resolution, so fewer pixels have to be reconstructed. Note how the reconstruction forms a 75 degree viewing cone, as the original images were taken with a 75 degree field of view.

Bibliography

[BBH03] M.Z. Brown, D. Burschka, and G.D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 25(8):993–1008, 2003.

[BE04] R. Bala and R. Eschbach. Spatial color-to-grayscale transformation preserving chrominance edge information. In *Proc. IS&T/SIDs 12th Color Imaging Conference*, pages 82–86, 2004.

[BRK10] M. Bleyer, C. Rother, and P. Kohli. Surface stereo with soft segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1570–1577. IEEE, 2010.

[Bru09] R. Brunelli. *Template matching techniques in computer vision: Theory and practice*. John Wiley & Sons Inc, 2009. ISBN: 978-0470517062.

[BVD98] E. Brenner and W.J.M. Van Damme. Judging distance from ocular convergence. *Vision Research*, 38(4):493–498, 1998.

[Cor06] Valve Corporation. Valve Map Format Documentation - The Valve Developer Community, 2006. `http://developer.valvesoftware.com/wiki/VMF_documentation` [Online; accessed 2-June-2010].

[DA02] U.R. Dhond and JK Aggarwal. Structure from stereo-a review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, 2002.

[Dou88] W.M. Doumas. Anti-aliasing of raster images using assumed boundary lines, October 25 1988. US Patent 4,780,711.

[FL95] P. Fua and Y.G. Leclerc. Object-centered surface reconstruction: Combining multi-image stereo and shading. *International Journal of Computer Vision*, 16(1):35–56, 1995.

[Fla10] M.T. Flanagan. Cubic Spline Interpolation (fast version), 2010. `http://www.ee.ucl.ac.uk/~mflanaga/java/CubicSplineFast.html` [Online; accessed 1-June-2010].

[FLP01] O. Faugeras, Q.T. Luong, and T. Papadopoulo. *The geometry of multiple images*. MIT press Cambridge, 2001. ISBN: 978-0262062206.

[GG99] S.A. Gabriel and K.E. Griffin. Method and system for texture mapping images with anisotropic filtering, December 21 1999. US Patent 6,005,582.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[HSC+05] S. Hrabar, G.S. Sukhatme, P. Corke, K. Usher, and J. Roberts. Combined optic-flow and stereo-based navigation of urban canyons for a UAV. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3309–3316. IEEE, 2005. ISBN: 978-0780389120.

[HWL03] A. Hirano, R. Welch, and H. Lang. Mapping from ASTER stereo image data: DEM validation and accuracy assessment. *ISPRS Journal of Photogrammetry and Remote Sensing*, 57(5-6):356–370, 2003.

[IDFF96] R. Ierusalimschy, L.H. De Figueiredo, and W.C. Filho. Lua-an extensible extension language. *Software Practice and Experience*, 26:635–652, 1996.

[K+93] A. Koschan et al. *What is new in computational stereo since 1989: A survey on current stereo papers*. Citeseer, 1993.

[KMS05] G. Krawczyk, K. Myszkowski, and H.P. Seidel. Lightness perception in tone reproduction for high dynamic range images. *Computer Graphics Forum*, 24(3):635–645, 2005.

[KTB89] D.J. Kriegman, E. Triendl, and T.O. Binford. Stereo vision and navigation in buildings for mobile robots. *IEEE Transactions on Robotics and Automation*, 5(6):792–803, 1989.

[LT94] R.A. Lane and N.A. Thacker. Tutorial: overview of stereo matching research. *Tina Memo*, 1:1–10, 1994. `http://www.tina-vision.net/docs/memos_vision.php#1994` [Online; accessed 17-November-2010].

[LTG+10] S.G. Latta, K. Tsunoda, K. Geisner, R. Markovic, D.A. Bennett, and K.S. Perez. Gesture Keyboarding, August 5 2010. US Patent App. 20,100/199,228.

[MF01] P. Moallem and K. Faez. Search space reduction in the edge based stereo correspondence. In *Proceedings of 6th International Fall Workshop on Vision, Modeling, and Visualization*, volume 2001, pages 423–429. Citeseer, 2001.

[MGM06] G. McTaggart, C. Green, and J. Mitchell. High dynamic range rendering in valve's source engine. In *ACM SIGGRAPH 2006 Courses*, page 7. ACM, 2006.

[MMF91] P.F. Mclauchlan, J.E.W. Mayhew, and J.P. Frisby. Stereoscopic recovery and description of smooth textured surfaces. *Image and Vision Computing*, 9(1):20–26, 1991.

[MP04] W. Matusik and H. Pfister. 3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. In *ACM SIGGRAPH 2004 Papers*, pages 814–824. ACM, 2004.

[New06] G. Newman. Garry's Mod, 2006. `http://www.garrysmod.com/` [Online; accessed 1-June-2010].

[NG10] L. Nalpantidis and A. Gasteratos. Stereo vision for robotic applications in the presence of non-ideal lighting conditions. *Image and Vision Computing*, 28(6):940–951, 2010.

[Nor07] A. Nordmann. Epipolar Geometry, 2007. `http://commons.wikimedia.org/wiki/File:Epipolar_geometry.svg` [Online; accessed 22-October-2010].

[OC89] G.P. Otto and T.K.W. Chau. A "region-growing" algorithm for matching of terrain images. *Image and Vision Computing*, 7(2):83–94, 1989.

[PP84] G.F. Poggio and T. Poggio. The analysis of stereopsis. *Annual Review of Neuroscience*, 7(1):379–412, 1984.

[Qia97] N. Qian. Binocular Disparity Review and the Perception of Depth. *Neuron*, 18(3):359–368, 1997.

[RG82] B. Rogers and M. Graham. Similarities between motion parallax and stereopsis in human depth perception. *Vision Research*, 22(2):261–270, 1982.

[SB95] J.P.P. Starink and E. Backer. Finding point correspondences using simulated annealing. *Pattern Recognition*, 28(2):231–240, 1995.

[Sch07] Scharstein, D. Middlebury Stereo, 2007. `http://vision.middlebury.edu/stereo/` [Online; accessed 24-June-2010].

[SD02] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring, March 26 2002. US Patent 6,363,170.

[SJBK08] J.K. Suhr, H.G. Jung, K. Bae, and J. Kim. Automatic free parking space detection by using motion stereo-based 3D reconstruction. *Machine Vision and Applications*, 21(2):163–176, 2008.

[SS02] 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.

[SS03] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1. IEEE, 2003. ISBN: 978-0769519005.

[ST02] G. Sharma and H.J. Trussell. Digital color imaging. *Image Processing, IEEE Transactions on*, 6(7):901–932, 2002.

[ST10] K. Shubina and J.K. Tsotsos. Visual search for an object in a 3d environment using a mobile robot. *Computer Vision and Image Understanding*, 114(5):535–547, 2010.

[Sto09] Stokkink, Q. StereoCam 2.7, 2009. `http://www.garrysmod.org/user/?u=71032` [Online; accessed 1-June-2010].

[Tay08] C.J. Taylor. Surface reconstruction from feature based stereo. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 184–190. IEEE, 2008. ISBN: 978-0769519504.

[Val04] Valve Corporation. Source Engine (2009 version), 2004. `http://source.valvesoftware.com/` [Online; accessed 1-June-2010].

[VN08] V. Vineet and PJ Narayanan. CUDA cuts: Fast graph cuts on the GPU. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

[Whe38] C. Wheatstone. "Contributions to the physiology of vision - Part the first. *On some remarkable, and hitherto unobserved, phenomena of binocular vision" Philosophical Transactions of the Royal Society of London*, 128:371–394, 1838.