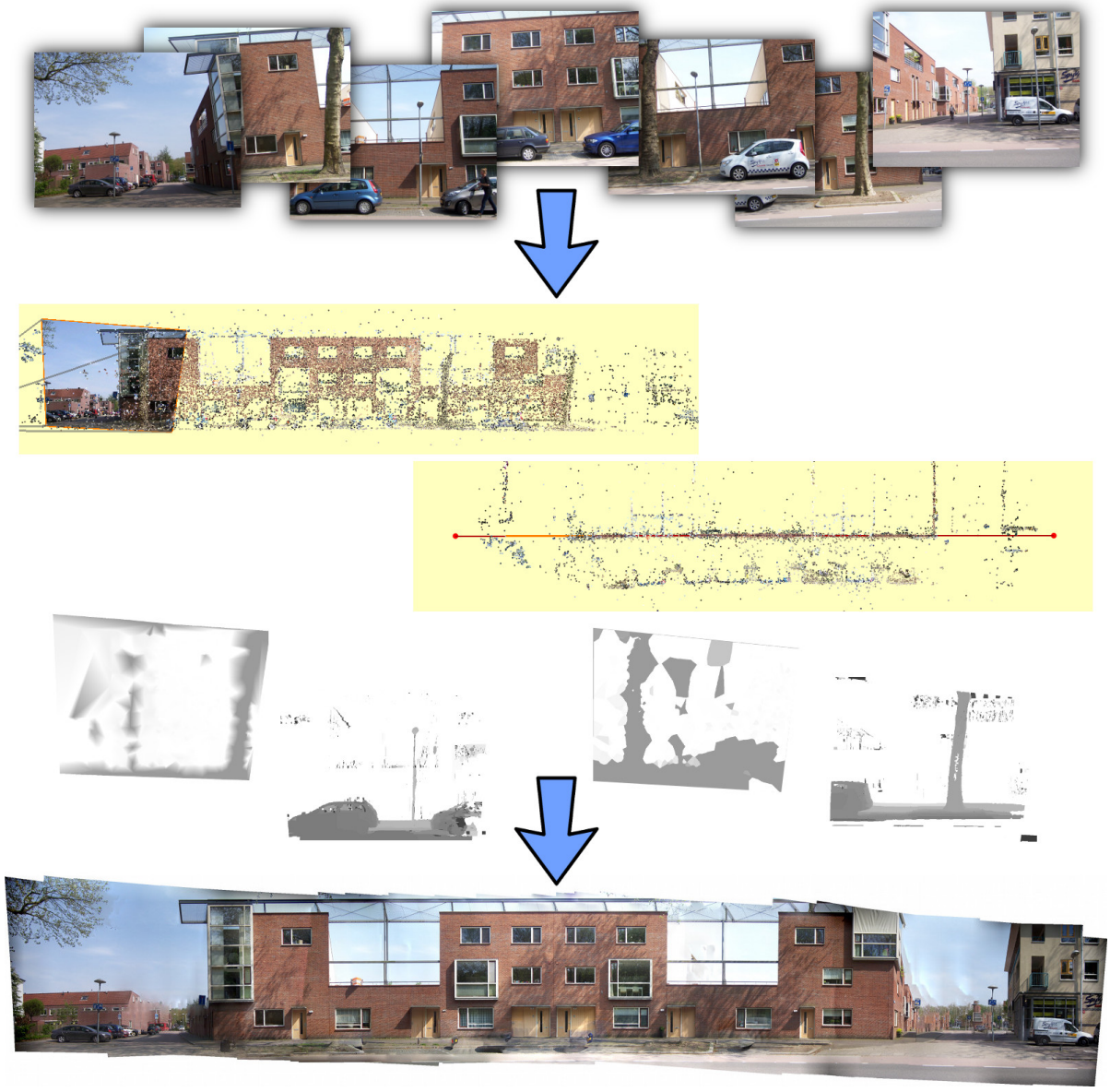
USING 3D INFORMATION TO AVOID FOREGROUND OBJECTS IN MULTI-VIEWPOINT PANORAMAS

BART VAN ANDEL

Master Thesis ICS-9908005



Department of Information and Computing Sciences
Faculty of Science
Utrecht University

August 2012 – version 1.0

Dedicated to the loving memory of my mother in law, who suddenly passed away of heart failure last June. She died at the age of 58.

Marjolijn Lugthart–Dolsma

25 August 1953 – 1 June 2012 †

ABSTRACT

In this master thesis, we introduce a proof-of-concept method to aid in foreground object removal for multi-viewpoint panoramas using both sparse and dense stereo information. The approach we extend from is able to generate seamless panoramas with these properties, but this often requires user intervention in an interactive refinement step. We try to reduce the amount of work the user has to perform by identifying and avoiding foreground objects automatically. We aim to assemble a seamless mosaic of a user-selected plane, while eliminating obstructing foreground objects as much as possible. To achieve our goal, we use Markov Random Field optimization to minimize a cost function, where one of the data terms is a new depth-based function which favors imagery close to or behind the picture surface. Depth information is inferred both from the sparse 3D point cloud generated in the initial reconstruction stage, and from stereo disparity maps computed by applying a dense stereo algorithm to the remapped source images. We demonstrate that our approach works for real images, and improves over the results of the method our research was extended from.

CONTENTS

ACRONYMS

LM	Levenberg-Marquardt
MVP	multi-viewpoint panorama
RANSAC	RANdom SAmpling Consensus
SfM	Structure-from-Motion
SVP	single-viewpoint panorama

1 INTRODUCTION

Multi-viewpoint panoramas (MVPs) are a type of panoramic image where multiple viewpoints are seamlessly combined into one image. Such panoramas are capable of displaying very wide scenes, like one whole side of a long street, or a large river bank.

For the creation of single-viewpoint panoramas (SVPs) – panoramas which are captured from a single viewpoint – many software applications exist. Software for creating MVPs on the other hand is virtually non-existent. The examples of MVP mentioned earlier at best have used the computer as a tool for the manual selection and blending of seaming locations.

For our graduation project, we have implemented a system which can produce a multi-viewpoint panorama from a set of images. This system is based on a 2006 article by Agarwala et al. [3]. Their goal is mainly to deliver visually pleasing panoramas. They do this by selecting the viewpoints that minimize an objective function, which is based on a set of desired image properties.

We wish to extend this system such that as much of the dominant surface is visible as possible by avoiding foreground objects. To this end, we have added an additional, depth-based data term to the objective function. This term penalizes viewpoints which contain objects in front of the picture surface.

Depth information is extracted from the scene in two ways. For each input photograph, we extract depth from the sparse 3D reconstruction which is computed in the preprocessing stage. We then triangulate the thus acquired sparse depth map by interpolating between the known values. We also extract depth by applying a stereo disparity algorithm to the projected images.

Our goal is to demonstrate that depth information extracted from the scene can be used to avoid foreground objects. We have successfully tested our implementation on a number of data sets, and in almost all cases, using our additional data term improves the results. An example of a successful application of our method can be seen in Figure 1.

Foreground object removal is useful when we are mainly interested in the scenery behind those objects. Real estate agents for instance will usually want to show properties to their potential customers, not in cars parked in front of those buildings, or even lamp posts and trees. Shopping center owners who want to use panoramas as a map to guide customers in the right direction likely also want a clutter-free

(a)

(b)

Figure 1: Demonstration of how well we succeed in foreground object removal using our additional data term on the GANSSTRAAT data set. (a) Using the STEREO-3 preset, nearly all foreground clutter is removed. (b) Result using the AGARWALA preset, which uses the same weights our predecessors use in their method.

view of their shops. Panoramas without foreground clutter could also be used in navigation applications, in which case we are interested in stationary objects, but not in dynamic objects like parked cars, which are essentially not a part of the street.

Interactive experiences like Google Street View [15] and Bing Maps Streetside [28] may benefit from large panoramas as well. Currently, both of these systems are centered around 360 degree panoramas taken at regular intervals. It is however not currently possible to get an overview of the side of a street in one image. Finding a particular building along a street requires virtually navigating across the street until the building is found, which takes effort (clicking around) and possibly a lot of time if the street is large. When large panoramas of the street were available, finding this building would take considerably less time, because a single image would show a much bigger part of the street than a single 360 degree photo could.

Microsoft has already conducted some research in that direction (Kopf et al. [22]). Their research focusses on an interactive view of the street, which combines multiple images into a single view on screen, although not as a highly detailed, blended panorama free from foreground clutter.

The remainder of this thesis is organized as follows. First we will provide some required background knowledge in chapter 2. In chapter 3, we will outline the terms used by the objective function, which is an integral part of the pipeline (chapter 4). Our results are presented and discussed in chapter 5 and 6. Finally, we conclude our thesis and provide some ideas for future work in chapter 7.

2 BACKGROUND

In this chapter, we will provide the most important background knowledge required to understand our method.

Section 2.1 will describe what a panorama is in the context of this thesis. The camera model used in our pipeline is detailed in section 2.2. Structure-from-Motion (SfM) will be explained in section 2.4. Stereo disparity is explained in section 2.3. Finally, section 2.6 deals with Markov Random Field optimization.

2.1 PANORAMA

The word *panorama* literally means "a complete view", and is a contraction from the Greek words πᾶν- *(pan-)*, meaning "all", and ὅραμα *(horama)*, from ὅραν *(horan)*, "to look, see". In the context of photography, one possible definition is this:

PANORAMA An image showing an unbroken view of a scene, with a very wide range of view.

We differentiate between single-viewpoint and multi-viewpoint panoramas.

2.1.1 *Single-viewpoint panorama*

Panoramas are often assembled from a series of images taken while rotating the camera around its own axes. So-called *stitching software* is then required to merge the images together into a seamless panorama, like PTgui[1], Hugin[2] or the Photomerge feature of more recent versions of Photoshop[3]. An example, stitched using Hugin, can be seen in figure 2.

1 http://www.ptgui.com
2 http://hugin.sourceforge.net
3 http://www.photoshop.com

Figure 2: A single-viewpoint panorama showing a full circular view of a bike path through some fields near Utrecht. Image by the author.

These days, many digital cameras and smartphones even supply some build-in kind of *panorama mode*, which can be used to create a panorama instantly, without having to resort to software. Examples include Sony's *Sweep Panorama*[4], Casio's *Slide Panorama*[5], and Nikon's *Easy Panorama Mode*[6]. Another approach is to use specially designed lenses (i.e., catadioptric lenses) to capture a full circular view of a scene in a single image.

All of these approaches have in common that they create panoramas with a single viewpoint, in other words, they are single-viewpoint panoramas (SVPs). Such panoramas can show up to a full spherical view of a scene around this viewpoint, and are very suitable for applications like virtual tours, especially when they are linked together using clickable *hotspots*. Well-known examples are Google Street View[7] [4] and its competitor, Bing Maps Streetside[8], as well as the 360Cities community[9]. They can also be remapped into a format suited for print, such as a stereographic projection, or the more recently developed Pannini projection [33]. Such images are very common and can be found, e.g., in the "panoramas"[10] and "360°"[11] groups at Flickr.

SVPs are less suited for very wide scenes, like a view of a whole street block, due to e.g. perspective distortion and the inability to deal with curves in the street.

Various approaches exist for creating a SVP. Some are based directly on the pixel data, others use features extracted from the pixels. Overviews are given by Chen and Klette [10] and, more recently, Szeliski [37]. As an example, Brown and Lowe use the following algorithm for recognising panoramas [8].

First, keypoints are extracted from the source images, which are then matched between images using a k-d tree. For each image, they then select m candidate matching images, and RANSAC is used to find geometrically consistence keypoint matches to find the homography between pairs of images. These image matches are verified using a probabilistic model. They then find connected components of images, which enables creating multiple panoramas at once in a set of unordered images. Then, for each connected component, bundle adjustment is applied to solve for rotation and focal length parameters of all cameras. Finally, the panorama is rendered by using multi-band blending.

---

4 http://www.sony.co.uk/article/id/1240556784154
5 http://di.casio.com/features/slide_panorama
6 http://www.nikonusa.com/Learn-And-Explore/Nikon-Camera-Technology/gjq8l3rt/1/Easy-Panorama-Mode.html
7 e.g., https://maps.google.com/?cbll=52.086,5.167&layer=c&cbp=13,-7,,0,-3
8 http://www.microsoft.com/maps/streetside.aspx
9 http://www.360cities.net/
10 http://www.flickr.com/groups/panoramas/
11 http://www.flickr.com/groups/360degrees/

2.1.2 *Multi-viewpoint panorama*

Multi-viewpoint panoramas (MVPs) can contain any number of viewpoints, and are suitable for very wide scenes, even containing curves. Creating a MVP is less straightforward than creating a SVP, and publicly available software capable of doing this is virtually non-existent. The process of assembling a MVP manually, while tedious and time-consuming, hasn't stopped people from creating a number of very interesting panoramas. Pioneering work in this area has been done by Thomas Riehle, who manually assembled large MVPs for his Ideal Views project [12]. Other notable examples can be found at the Seamless City[13] and PanoramaStreetline[14] websites.

Since we focus on MVPs in this thesis, we will simply refer to them as "panoramas" in the rest of this work.

The basic outline for creating an MVP, as implemented by our system, is as follows.

The system takes an unordered set of input photographs, which are first processed by lens correction software to remove radial distortions. Structure-from-Motion (SfM) is used to obtain the camera properties as well as a sparse 3D point cloud representing the scene. A simple brightness correction step is applied to account for exposure differences using point matches between the input photographs. The user then defines a picture surface by drawing a line in a top-down view of the scene, which is extruded in the y direction. The system projects the photographs this surface and computes a labeling for viewpoint selection automatically, based on a set of desired properties. Finally, a seamless panorama is created by applying this labeling to the projected images and blending them using Laplacian blending.This is done to smooth away errors left by exposure differences or mismatching scenery.

Chapter 4 will give a more detailed version of the pipeline used by our system, which implements this approach.

2.2 CAMERA MODEL

A camera (comprising of a camera body plus attached objectives, filters etc.) can be described mathematically by a set of parameters.In our system, we use a pinhole camera model, which assumes a rectilinear projection in the source images; lines which are straight in the real world, will remain straight in the camera image.

A camera is called calibrated if its parameters are known; otherwise, the camera is called uncalibrated.

12 http://www.tomas-riehle.de/index_en.php
13 http://www.seamlesscity.com/
14 http://panoramastreetline.com/

The camera parameters are devided into two different components: the intrinsic and the extrinsic parameters.

2.2.1 *Intrinsics*

The intrinsic parameters of a camera define the photographic properties regardless of its location and orientation. Both linear and nonlinear components can be present. Focal length, image format, and principle point are linear components, while lens distortion parameters generally are nonlinear.

The linear components can be encoded in a matrix:

$$A = \begin{pmatrix} \alpha_x & s & u \\ 0 & \alpha_y & v \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Here, $\alpha_x = f \cdot m_u$ and $\alpha_y = f \cdot m_v$ represent the focal length in terms of pixels, where (m_u, m_v) is a scale factor which converts real-world distances to pixels (i.e., pixels per mm). s is a skew coefficient between the x and y axis. Typically cameras use square pixels with no skew present between both axes, in this case $m_u = m_v$ and $s = 0$. The principal point $(u, v) = (m_u t_u, m_v t_v)$ is ideally located in the center of the image: $(u, v) = (0, 0)$. NB: In our system, f will be used to denote focal length in pixels.

Non-linear components, e.g. lens distortion parameters, cannot be encoded into this matrix. There are several ways to deal with lens distortion parameters. In our system, we use a simple polynomial radial distortion function using two parameters k_1 and k_2:

$$r(\mathbf{p}) = 1 + k_1 \|\mathbf{p}\|^2 + k_2 \|\mathbf{p}\|^4 \tag{2}$$

Here, p is the normalized distance of a pixel to the center of the camera.

2.2.2 *Extrinsics*

The extrinsic parameters R and $\mathbf{t}$ respectively define the rotation and translation of the world coordinate system with respect to the camera center. To find the rotation and translation of the camera in world coordinates, we have to apply the reverse transform:

$$C = R^{-1}\mathbf{t} \tag{3}$$

Because R is a rotation matrix, this is equivalent to:.

$$C = R^{T}\mathbf{t} \tag{4}$$

To project a 3D point $\mathbf{X}$ from world coordinates into a camera, we use the following procedure:

1. Convert from world coordinates to camera coordinates:

$$\mathbf{P} = R\mathbf{X} + \mathbf{t}$$

2. Convert to 2D by applying perspective division:

$$\mathbf{p} = -\mathbf{P}/\mathbf{P}.z$$

3. Convert to pixel coordinates:

$$\mathbf{p}' = f \cdot r(\mathbf{p}) \cdot \mathbf{p}$$

2.3 STEREO CORRESPONDENCE

Stereo correspondence is a fundamental problem in computer vision. Using stereo disparity algorithms it is possible to reconstruct a depth (or disparity) map from a scene using 2 or more images. The 2-view case is often referred to as *binocular vision*, after the way humans see.

Using more than 2 views usually leads to a more robust depth estimation because errors can be identified and smoothed out more easily. Also, it can handle occlusions better due to the availability of more alternative viewpoints.

Zhang et al. describe an algorithm for the robust recovery of consistent depth maps from video. Their approach is targeted at video, but they mention it also works when there are large baselines between the frames, which is the case with our photographic input. Based on the results displayed in the article and at their website, it is likely that their approach will generate very consistent depth maps.

We have opted for a much simpler approach, by computing depth maps only for pairs of neighboring images. We use a graph-cut based stereo disparity algorithm by Kolmogorov and Zabih [21], as it is implemented in OpenCV.

We realize that this algorithm is not state-of-the-art. In the Middlebury Stereo Evaluation [30, 32], it only achieves a ranking of 80.4 (as "GC+occ") on their test data sets, whereas the currently top ranked algorithm has a ranking of 8.6. However, it is sufficient for the purpose of demonstrating that stereo disparity can be used successfully for the removal of foreground objects.

2.3.1 *Stereo disparity*

Disparity can be extracted from a pair of images by finding corresponding pixels for the pixel values of one image in another image and computing their distance, in pixels. If the baseline B between the cameras is known, the depth Z can now be computed from the disparity d and focal length f using the following formula:

$$Z = f\frac{B}{d} \tag{5}$$

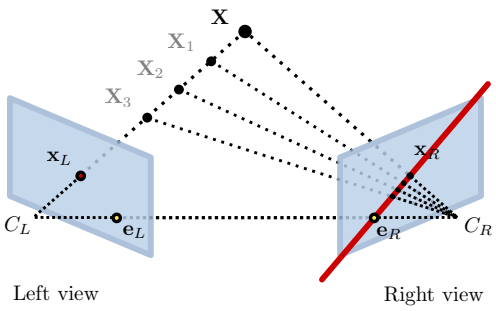


Figure 3: Epipolar geometry. A point $\mathbf{X}$ observed by camera C_L at $\mathbf{x}$, must lie on a line as observed by camera C_R, as defined by the epipolar constraint. The baseline B is the length of the virtual line connecting the camera centers C_L and C_R. The epipolar plane is formed by C_L, C_R, and the observed point $\mathbf{X}$. The red line is the epipolar line $e(\mathbf{x_L})$, which depends on the position of $\mathbf{x_L}$, and lies in the epipolar plane.
Original image by Arne Nordmann (adapted to our needs).

2.3.2 *Epipolar rectification*

Without rectification, the search space for finding corresponding pixels is essentially the whole image. Epipolar rectification effectively reduces the problem to a 1-dimensional search problem, which also decreases the likelihood of false matches. Most stereo algorithms assume rectified images as input.

Figure 3 visualizes the epipolar geometry. The epipolar constraint states that for a 3D point $\mathbf{X}$ projected into camera C_i at $\mathbf{x}_i$, the corresponding point $\mathbf{x}_j$ must lie on the *epipolar line* $e(\mathbf{x}_i)$ in camera C_j. The epipolar line is dependent on the position of the projected point $\mathbf{x_i}$. For other points the line will be different, but all epipolar lines will intersect in a single point, known as the *epipole*.

Epipolar rectification transforms both images such that the epipolar lines become collinear and parallel to the x-axis. Both epipoles now lie at infinity. Assuming a rigid scene, after rectification projected points of the same world point will always lie on the same horizontal line.

There are multiple ways to find the transformation parameters required for epipolar rectification. Often these involve the fundamental matrix (e.g. Mallon and Whelan [26]), which describes the epipolar geometry between two cameras.

2.3.3 *Fundamental matrix*

The fundamental matrix $F_{i,j}$ is a 3×3 matrix which relates corresponding points in pairs of stereo images I_j and I_j. When using homogeneous image coordinates $\mathbf{x}_i$ and $\mathbf{x}_j$ of a point match between

images I_i and I_j, $F_{i,j}\mathbf{x}_i$ defines an epipolar line on which the corresponding point $\mathbf{x}_j$ must lie in the other image, i.e.:

$$\mathbf{x}_j^T F_{i,j} \mathbf{x}_i = 0 \tag{6}$$

The fundamental matrix can be estimated robustly by solving the 8-point algorithm (Hartley [17]) using point matches inside a RANdom SAmpling Consensus (RANSAC) procedure. It contains information about its two epipoles and about the homography between the two pencils of the epipoles.

2.3.4 *Essential matrix*

The essential matrix E is a 3×3 matrix which, just like the fundamental matrix, relates corresponding points in pairs of stereo images. The only difference between the two is that the fundamental matrix deals with uncalibrated cameras, whereas the essential matrix can only relate calibrated cameras.

If the camera extrinsics are known, the essential matrix E can be computed from the fundamental matrix:

$$E_{i,j} = A_j^T F_{i,j} A_i \tag{7}$$

Here, A_i and A_j contain the camera intrinsics for images I_i and I_j, respectively. Conversely,

$$F_{i,j} = A_j^{-T} E_{i,j} A_i^{-1} \tag{8}$$

The essential matrix carries information about rotation R and the direction of translation $\mathbf{t}$ (that is, the translation up to a scale). By defining $[\mathbf{t}]_\times$ as the matrix such that $[\mathbf{t}]_\times \mathbf{x} = \mathbf{t} \times \mathbf{x}$ for any vector $\mathbf{x}$, this can be written as:

$$E = [\mathbf{t}]_\times R \tag{9}$$

Other properties of the essential matrix are that it has rank zero, and its to non-zero determinants are equal.

2.4 STRUCTURE-FROM-MOTION

Structure-from-Motion (SfM) [18] is an approach to simultaneously recover camera parameters (both intrinsics and extrinsics) as well as a sparse 3D reconstruction representing the scene from an unordered set of images, up to a scaling factor. Many different implementations of SfM systems exists, tailored at different problems.

Früh and Zakhor use SfM to create virtual cities by aligning photographs of urban scenes captured by car- and aeroplane-mounted cameras into 3D models [13, 14]. Snavely et al. have created an interactive viewer for virtually exploring popular scenes of the world,

where SfM is used to create large 3D models from community photographs [35, 36]. For his master thesis, Broere [7] has researched the feasibility of using SfM for urban scene reconstruction for the purpose of navigation.

We use the Bundler software package developed by Snavely [34, 35], which implements a number of steps. In the first stage of the process, correspondences between the images are found. The second stage uses an iterative bundle adjustment approach to find the camera parameters and generate a sparse 3D reconstruction of the scene.

2.4.1 *Finding correspondences*

First, SIFT keypoints [25] are extracted from the images, which are designed to be distinctive and invariant to translation, rotation and scale. Furthermore they are robust against global illumination differences and noise, and it can handle small amounts of tilt.

The SIFT points are then matched between all pairs of images using an approximate nearest neighbor search, and for each pair a fundamental matrix is estimated using RANSAC. Outliers to these recovered fundamental matrices, i.e. mismatched points, or points that lie on objects that were moved, are removed. We can now count the number of matches between each image pair and to get an estimate of to what degree they are connected.

Figure 4 visualizes two such correspondence matrices, for our GANSSTRAAT data set versus our ADELAARSTRAAT2 data set. The former shows a much stronger connection between the images than the latter, in this case because they were taken with much more overlap between the photographs.

After this step the point matches are organized into tracks, which are connected sets of matching keypoints across multiple images. Tracks which contain more than 1 keypoint from the same image are rejected as inconsistent. The remaining tracks which contain at least 2 keypoints are used as input for the reconstruction phase.

2.4.2 *Iterative bundle adjustment*

Next, a minimization problem is formulated which recovers a set of camera parameters and a 3D location for each track by minimizing the reprojection error of reconstructed 3D points back into the cameras. This minimization is formulated as a non-linear least squares problem, which will be solved using sparse bundle adjustment.

Starting with a single pair of images, the minimization is computed iteratively by adding one or a few images at a time, until no remaining cameras observe any reconstructed 3D points. Each iteration, the image is added which observes the most data points M already in

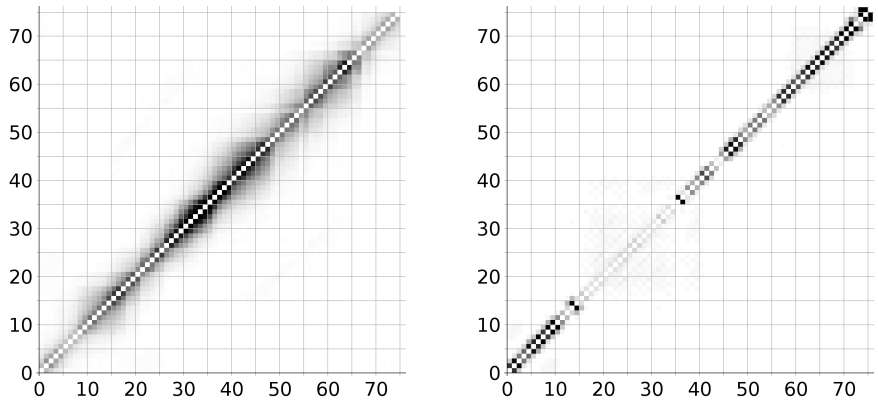


Figure 4: Correspondence matrices for our GANSSTRAAT and ADELAARSTRAAT2 data sets after RANSAC outlier filtering, displayed as grayscale images. Axes represent image numbers. Darker values indicate a higher number of keypoint matches between the pair (values capped to 2000). Images are never matched against themselves.

the optimization, and additionally, every other image with at least 0.75M matches to existing points. For each of these cameras, their extrinsics and intrinsics are estimated using a Direct Linear Transform (DLT) technique inside a RANSAC procedure. Tracks of the newly added images are added if they are observed by at least one other reconstructed image, and their triangulated 3D location is well-conditioned. After each iteration, an outlier filtering step is executed which removes points having too high a reprojection error in the current solution.

SfM algorithms are prone to getting stuck in bad local minima with degenerate reconstructions, so it is important that the algorithm is bootstrapped with images that minimize this chance. The initial pair of images is therefore chosen such that they have as much corresponding points as possible, while they cannot be described by a single homography.

2.5 RANDOM SAMPLING CONSENSUS (RANSAC)

When fitting an unknown model to a set of known data points using e.g. a least-squares method like Levenberg-Marquardt (LM), outliers (noise points) can prevent the algorithm from finding an optimal solution. This can be overcome by using RANSAC [12].

The idea of the RANSAC algorithm is to iteratively select a random subset of the data points as *hypothetical inliers*. After fitting a model to this subset using the original algorithm (e.g., LM), all other data points are tested against the computed model. Points which correspond to the model (up to a given factor) are added to the set of

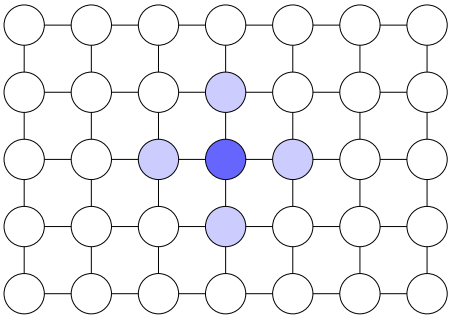


Figure 5: Markov Random Field, represented as a graph where every vertex is connected to 4 neighbors (except for vertices at the borders).

hypothetical inliers. When enough points were classified as inliers, the model is considered good. If too few points were considered correct, the model is rejected. Otherwise, using the new set of inliers, the model is reestimated, after which the model is evaluated by computing the error of the inliers to the model.

This procedure is repeated a fixed number of iterations, keeping the best model in each iteration. After the last iteration, the best model (if any) is returned as the solution. Optionally the algorithm can quit earlier if a sufficiently good solution has been found according to a specified maximum allowable error.

2.6 MARKOV RANDOM FIELD

A *Markov chain* is a collection of random variables X_t, where the future state of the variables is conditionally independent of the past state: each state is only dependent on the state immediately preceding that state.

When this concept is applied to a lattice (such as an image I), we can speak of a Markov Random Field (MRF). In this case, the state of each pixel $p = I(x, y)$ is only dependent on its first-order neighbors $\mathcal{N}$. Commonly a 4-neighborhood is used, for which $\mathcal{N} = I(x, j-1), I(x+1, j), I(x, j+1), I(x-1, j)$. This is visualized in Figure 5.

MRFs can be applied to solve a wide variety of problems, including but not limited to image noise reduction [16, 27], inpainting [29] and texture synthesis [23, 24]. An early overview of applications for MRF is given by Kindermann and Snell [20]. We used it for optimizing our objective function to determine the viewpoint selection labeling.

2.6.1 *Energy minimization*

A common problem is to find some labeling L, which assigns to each pixel $p \in I$ a label $L(p)$, such that an energy function is minimized. The general form of such an energy function is given by:

$$E(L(p)) = \sum_{p \in I} D(p, L(p)) + \sum_{p,q \in \mathcal{N}} V(p, L(p), q, L(q)), \tag{10}$$

A comparative study of energy minimization methods for MRFs can be found in Szeliski et al. [38]. We use a graph-cut based method by Boykov et al. [6], which minimized the energy function in a number of α-expansion moves.

2.7 PLANAR SUBDIVISION

In our application we use Voronoi diagrams and Delaunay triangulations to interpolate pixel values based on known values for a subset of those pixels.

2.7.1 *Delaunay triangulation*

A Delaunay triangulation $DT(\mathcal{P})$ is a triangulation of a set of points $\mathcal{P}$ such that no point $p \in \mathcal{P}$ is inside the circumcircle of any triangle in the triangulation. Hence, for any given point $p \in \mathcal{P}$, its nearest neighbors $n \in \mathcal{P}$ can be found by following the edges for which p is an end point. For any point x, the nearest points $p \in \mathcal{P}$ are the vertices of the Delaunay triangle in which it is contained.

2.7.2 *Voronoi diagram*

The Voronoi diagram is the dual graph of the Delaunay triangulation $DT(\mathcal{P})$. It consists of convex, polygonal cells around the points $p \in \mathcal{P}$, in such a way that for all points x within a site P around p, the nearest point from $\mathcal{P}$ is p:

For image I and a set of defined points $\mathcal{P}$, the Voronoi regions R_k around each point $p_k \in \mathcal{P}$ (with $k \in K$ the set of indices), are defined by

$$R_k = \{x \in I \mid \forall j \neq k : d(x, P_k) \leqslant d(x, P_j)\} \tag{11}$$

where P is the site around p, and $d(x, A) = \inf\{d(x, a) \mid a \in A\}$ denotes the distance between the point x and the subset A.

The Voronoi diagram is the tuple of all cells $(R_k)_{k \in K}$.

3 METHOD

The system described by Agarwala et al. [3] aims to create a visually pleasing, continuous, multi-viewpoint panorama with depth cues still present. This chapter focusses on the cost function which is optimized to determine the labeling for viewpoint selection. This labeling is used at the end of the pipeline to determine for each pixel which input photograph to use. A complete overview of the system as we have implemented it is given in chapter 4.

This chapter is organized as follows. First we will describe the desired image properties. Our cost function is basically a mathematical representation of these properties. Then, we will explain each part of the cost function individually, one section per term.

3.1 DESIRED IMAGE PROPERTIES

We want our panoramas to have the following properties:

1. To keep perspective distortion to a minimum, every object in the scene is rendered from a viewpoint roughly in front if it.
2. The panorama consists of large parts of linear perspective. The images from which the panorama is composed are taken from viewpoints where a person would normally stand. E.g., a person would normally view a city block from across the street, rather than from a faraway viewpoint.
3. The panorama exhibits local perspective clues. Objects closer to the camera appear bigger, and multiple vanishing points can be seen.
4. The panorama appears continuous and natural. There are no visual seams between regions originating from different source images.
5. *new* The picture surface selected by the user is clearly visible; cluttering foreground objects are eliminated from the view.

The first 4 properties are adopted from Agarwala et al. [3]. The 5th property describes what we additionally want with the panorama.

We realize that we will not always be able to create an image which adheres to all of these properties. For example, we cannot render the picture surface uncluttered at image locations where no photographic source data is available which looks around the cluttering object. This

is especially likely to happen when objects are very close to the picture surface compared to the viewpoints the source photographs were taken from, or when the objects are very big, as visualized in Figure 10. Depending on the complexity of the scene, this problem may be solved using inpainting techniques, which we have not implemented in our implementation of the system. To look around big objects, it is also required that the restriction on perspective distortion from skew viewpoints is relaxed a bit. This can be accomplished easily by changing the weight of a particular data term in the cost function.

The overal cost function which we try to minimize is as follows:

$$\begin{aligned}\sum_{p}\Big(\quad & \alpha && D(p, L(p)) \\ & +\beta && H(p, L(p)) \\ & +\gamma_v && Z_v(p, L(p)) \\ & +\gamma_d && Z_d(p, L(p)) \\ & +\gamma_s && Z_s(p, L(p))\Big) \\ +\sum_{p,q} & && V(p, L(p), q, L(q))\end{aligned} \tag{12}$$

where α, β, γ_v, γ_d and γ_s are parameters which can be tuned to influence the weight of each component. p is the pixel we are currently looking at, $L(p)$ is the labeling for this pixel, and q is a neighboring pixel.

D, H, Z_v, Z_d and Z_s are data terms, and V is the smoothness prior, each of which is detailed in the following subsections.

Not part of the cost function is a guard which prevents the optimizer from selecting viewpoints where the data is *null*, i.e. pixels outside the field of view of the camera, where no data was captured. We do enforce this constraint is in our optimizer though by not allowing $L(p) = i$ if $I_i(p) = \textit{null}$.

3.2 IN-FRONTNESS

We want perspective clues to show up in our final panorama. However, we want to avoid viewpoints which are distorted too much. Perspective distortion occurs wherever the camera does not point straight at the surface, an example of which is given in Figure 6. Therefore, we prefer views from cameras pointing straight at the picture surface, where perspective distortion is minimal, over views taken from a considerable angle.

To satisfy this desided property, we define a cost term, which adds a penalty to views not pointing straight at the scene. For each camera C_i, we find the pixel p_i whose corresponding 3D sample $S(p_i)$ lies closest to the camera. This is the most straight-on view possible for

Figure 6: Comparison of (a) a straight viewpoint and (b) a viewpoint from an angle of the same part of a scene. When included in a wide panorama, the stairs appear more natural when depicted from the straight viewpoint.

this camera. The cost for point p is then simply defined as the 2D distance to p_i:

$$D(p, L(p)) = |p - p_{L(p)}| \tag{13}$$

This is computationally cheaper than using the real angle between the surface and the line defined by connecting C_i to $S(p_i)$. As long as the distance of each camera to the picture surface is roughly the same, our approximation works well enough.

3.3 RESEMBLE AVERAGE

Assuming that the color channels vary in value from 0-255, we use the following term to favor viewpoints which resemble the average for those pixels with a low standard deviation:

$$H(p, L(p)) = \begin{cases} |M(p) - I_{L(p)}(p)| & \text{if } \sigma(p) < 10 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Here, σ is the medium absolute deviation (MAD) from the median image $M(p)$. The constant with a value of 10 is adopted from Agarwala et al.. They do not specically address the value of this constant but we assume they have found it empirically.

The MAD is computed as the median L_2 distance from the median color, which again is computed robustly using a Vector Median Filter [5] across the red, green and blue image channels.

The median is preferred over a simple average because it is less sensitive to outliers. In our application, this means that a single image which does not agree with the other images on the color of a certain pixel is simply left out. Needless to say this will only work if there are 2 or more alternative viewpoints which do agree on the color.

This cost term was added as a simple means to remove e.g. people walking across the scene. However, this does not work very well when our object of interest is blocked from view in more images than where

it is visible. For instance, suppose we see the wall of a building in 2 images, while it is blocked by a car parked in front in 5 images. In that case, the pixel is more likely to get the color of the car, instead of the wall.

3.4 FOREGROUND OBJECT REMOVAL

The problem we try to address is the automatic removal of foreground objects from the final panorama. We will try to do this using data from the input photographs directly, that is, without using cloning or inpainting techniques. Our approach tries to avoid foreground objects by routing the seams around those objects.

We identify foreground objects by computing a depth map Z_i, for each projected photograph. The depth map contains, for each output pixel p, a signed value indicating the distance between the picture surface and the 3D point $S(p)$ corresponding to that pixel. Negative values indicate that the 3D point is located behind the picture surface (as seen from the camera).

Because we are interested in removing objects in front of the picture surface, but do not care about objects behind it, we omit negative depths by capping the negative values to zero. Our final data term then becomes:

$$Z(p, L(p)) = \max\{0, Z_{L(p)}(p)\} \tag{15}$$

We have explored two different techniques to infer depth. Section 3.4.1 deals with depth information extracted from the 3D reconstruction computed as part of the panorama pipeline. In section 3.4.2 we show how we use stereo disparity to extract depth information from the projected photographs

3.4.1 *Sparse Depth from 3D Reconstruction*

During the preprocessing stage of the panorama creation pipeline (see Chapter 4), we compute a sparse 3D reconstruction of the scene captured by the input photographs. The reconstruction contains for each camera C_i its estimated focal length f, a rotation matrix R, and a translation vector $\mathbf{t}$. It also contains a sparse cloud of reconstructed 3D points X_j, together with corresponding 2D point locations $x_{i,j}$ in the source photographs I_i they were reconstructed from.

We assume that the reconstruction contains no wrong points. We create a sparse depth map for each source photograph by computing the signed shortest distance of each 3D point X_j observed by camera C_i to the picture surface S and assigning it to the corresponding 2D pixel at $x_{i,j}$.To create a complete depth map, we need to interpolate between the sparse depth values somehow.

Two interpolation methods have been examined, both based on a planar subdivision of the source photograph based on the points $x_{i,j}$. When the reconstruction is very dense (i.e. lots points have been generated, leaving little area uncovered), those methods will result in similar depth maps. In practice however, we most reconstructions are pretty sparse, resulting in large differences between the generated depth maps.

Regardless of the interpolation method, $z(p)$ is simply the shortest distance of point p (whether observed or interpolated) to the picture surface S.

3.4.1.1 *Delaunay*

Using a Delaunay triangulation of our points $x_{i,j}$, $DT(x_{i,j})$, we can fill in the unknown pixels by linearly interpolating between the depth values of the known pixels $p \in \mathcal{P}$. This results in a smooth depth map. We use the Delaunay triangulation because of its property that the closest vertices to any point p can be found by looking up the triangle which contains the point.

Here, we assume that the real geometry is approximately smooth between the known points.

3.4.1.2 *Voronoi diagram*

We use the Voronoi diagram as an even simpler means of interpolation, by assigning all points $x_{i,j}$ within the cell around p the depth value of p. This results in a depth map with strongly defined regions. When a point p is contained within a Voronoi region R_k, than the single closest point to p is P_k.

The rationale behind this is similar to the one used for the Delaunay triangulation, except it better resembles geometry with more abrubt changes in depth, rather than smooth changes.

3.4.2 *Dense Depth from Stereo Disparity*

Instead of relying on the sparse 3D reconstruction to generate a depth map, we can use a stereo disparity algorithm to compute a dense depth map from the actual image data. Stereo disparity, being a fundamental subject in computer vision, has been extensively studied, resulting in a whole encyclopedia of different algorithms (see e.g. the Middlebury Stereo Page [31]) Because so many algorithms have been developed, it is not easy to choose an algorithm which works best in all our cases. And because we just want to show a proof-of-concept, we have picked one based on availability.

The algorithm we use is a graph-cut based algorithm by Kolmogorov and Zabih [21]. One of the reasons why we have chosen this algorithm is because it is implemented in OpenCV, a library we

already had experience with. The two other algorithms in OpenCV – Block Matching (BM) and Semi-Global Block Matching (SGBM) – while faster by several orders of magnitude, performed considerably worse on the data we tested the algorithms with. Overall, the graph-cut based method gives crisp results with well-defined edges.

Stereo algorithms generally require that the images be rectified. As outlined in section 2.3, this is usually accomplished by finding the epipolar geometry. Because we already have the projected images on a common plane, under a few assumptions we can use this directly to acquire depth. We assume that the camera was kept at a steady height with respect to the picture surface, and that the distance to the picture surface is approximately constant.

A violation of the first assumption could be addressed by simply rotating the warped images in 2D such that the y components of the camera positions would line up again. To fix violations to the second assumption however would require resorting to epipolar geometry altogether.

For simplicity, we further assume that the images were taken while advancing strictly in the same direction, i.e. not switching back and forth. Finally, 2-view stereo correspondence can only handle cameras with a baseline > 0. Therefore, we currently do not support multi-row panoramas, where multiple image were captured from each camera viewpoint.

For our data sets we have captured the photographs in a way that corresponds to all these assumptions and can thus use this simplified approach.

Since we are not interested in disparity, but in depth, we convert the disparity values to a depth value, which in our case is defined as the distance of the objects to the picture surface. When the baseline is constant, we can use the following formulas to convert disparity:

$$Z_{\text{left}}(p) = d_C\left(1 - \frac{T}{T - Z_{d,\text{left}}(p)}\right) \tag{16}$$

$$Z_{\text{right}}(p) = d_C\left(1 - \frac{T}{T + Z_{d,\text{right}}(p)}\right) \tag{17}$$

where d_C is the distance of the cameras to the picture surface S, T is the baseline between the cameras, and $Z_d(p)$ is the disparity computed for pixel p.

For a variable baseline, the formulas become slightly different:

$$Z_{\text{left}}(p) = d_C\left(1 - \frac{T_{\text{left}}}{T_{\text{left}} - Z_{d,\text{left}}(p)}\right) \tag{18}$$

$$Z_{\text{right}}(p) = d_C\left(1 - \frac{T_{\text{right}}}{T_{\text{right}} + Z_{d,\text{right}}(p)}\right) \tag{19}$$

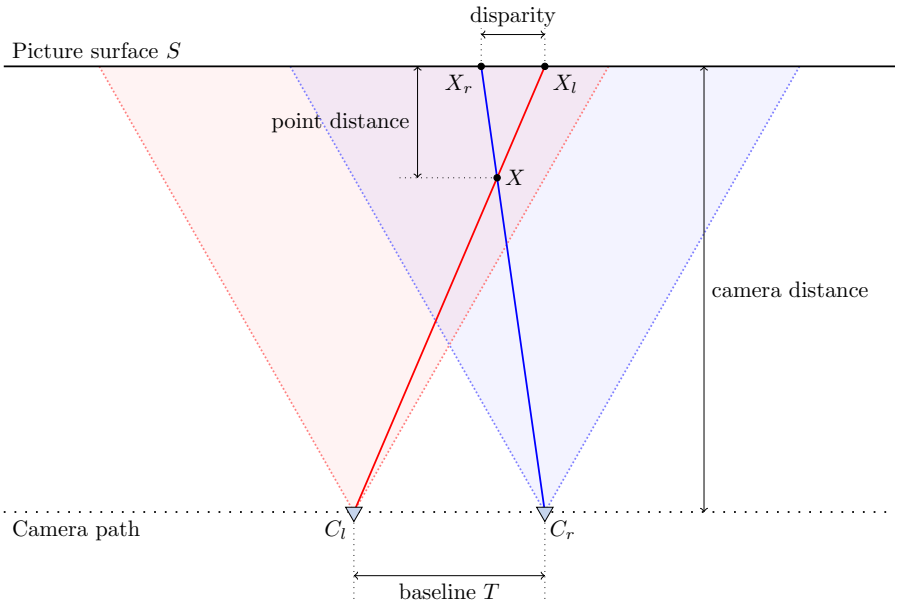


Figure 7: Computing the distance of a point X to the projection surface S using stereo disparity. This figure shows an xz cross-section of a virtual scene. We want to compute the distance of a point X to the projection surface S. Point X is projected at X_l from camera C_l, and at X_r from camera C_r. The disparity between those projected points is used to compute the orthogonal distance of X to the projection surface S.

with $T_{\text{left}} = |C_{L(p)} - C_{L(p)-1}|$ and $T_{\text{right}} = |C_{L(p)+1} - C_{L(p)}|$, assuming the images are numbered in increasing order from left to right.

Each neighboring pair of input photographs generates two disparity images: one for the left camera and one for the right. In other words, for all but the the leftmost and rightmost input photographs, two depth images are generated. Z_{left} was generated when the image was used as the left camera, and $Z_{s,\text{right}}$ when it was used as the right camera.

We combine the left and right depth map by taking for each pixel the maximum of the two:

$$Z(p) = \max\{Z_{\text{left}}(p), Z_{\text{right}}(p)\} \tag{20}$$

See figure 7 for a graphical explanation.

Because occlusions are unfortunately returned as zero disparity by the software we use, we cannot take a very sophisticated approach when it comes to merging them into one. Ideally, we would differentiate between real depth information and occlusions, but because we can't, we simply take the maximum value of both images. In practice this turned out to be a viable solution, the combined depth map aligned up with the scene pretty well upon inspection.

3.5 SMOOTH TRANSITIONS

It is shown by Kwatra et al. [23] and Agarwala et al. [1] that a seamless transition will minimize the following function:

$$V(p, L(p), q, L(q)) = \|I_{L(p)}(p) - I_{L(q)}(p)\| + \|I_{L(p)}(q) - I_{L(q)}(q)\| \quad (21)$$

for each neighboring pairs of pixels p and q, where $\|\cdot\|$ denotes an appropriate norm. We use the L_2 norm by default, after Agarwala et al., but allow this to be changed.

4 IMPLEMENTATION

The proposed method is integrated into a panorama creation pipeline, the outline of which is shown in Figure 8.

Our pipeline is based on Agarwala et al.'s pipeline, but deviates from theirs in a number of essential parts.

We optimize our MRF directly on the finest level, at the full resolution of our output panorama. Agarwala et al. use a hierarchical way to optimize the MRF [2], which first optimizes the MRF at a coarse level, and then refines the labeling thus found by scaling it to full size and re-optimizing only in the neighborhood of the seams. This is typically a lot faster and less memory intensive than a non-hierarchical approach which operates only at the finest level. Because of this, it scales better with larger images. However, the property that the minimum found is provably close to the global minimum is lost.

Their pipeline contains an interactive refinement step, which we have skipped because it is of no value for our research. For a finished product this would be a valuable addition, because there is not guarantee that the system will provide a visually optimal panorama.

Finally, the gradient domain blending step has been replaced with a more simple Laplacian blending. Gradient domain blending works on the 1st derivative of the images and requires solving the Poisson equation. Laplacian blending, while often visually inferior to gradient domain blending, is more easy to implement. More importantly, we found an implementation (multiblend [19]) which we could use for our research without much modification, except for allowing it to use our labeling. We added this step purely for aestetics, again because for our research seam placement is far more important than seam blending.

Our pipeline is build from a number of components, which are described in the next few sections. For the offline parts of the pipeline, we have created a number of batch processing scripts to make trying different parameter sets more easy.

In section 4.1 we describe the acquisition stage. Section 4.2 deals with the preprocessing stage. Section 4.3 shows how the picture surface is selected by the user. The setup of the cost function is described in section 4.4, and the computation of the viewpoint selection labeling will be handled in section 4.5. Finally, section 4.6 describes how we apply the labeling to assemble our output panorama.

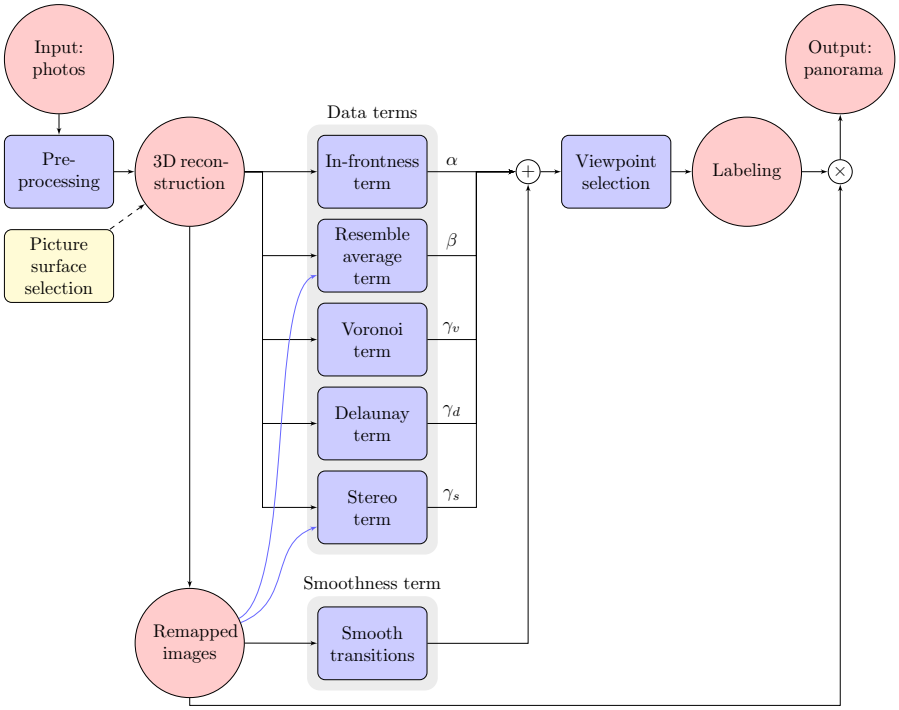


Figure 8: Flowchart depicting the pipeline implemented by our system. Red circles represent input, intermediate and output data. Blue rectangles are system actions, and yellow rectangles are user actions.

4.1 ACQUISITION

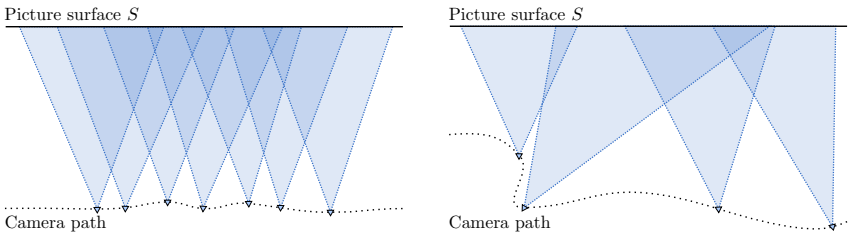
Although not technically a part of the pipeline, the way the source photographs are acquired has a huge influence on how our system will perform.

As with many other systems, the *garbage in, garbage out* principle also applies to our system. The SfM step requires a decent amount of overlap between the input images. Additionally, our stereo step requires the images to be taken at a constant height and preferably also with a constant distance between shots. Therefore, we must follow a few simple rules when capturing our source photographs to increase the chance of a successful panorama creation.

We want our photographs to be taken at regular intervals, while keeping unwanted variations to a minimum. We do this by walking along a line parallel to the scene, capturing a properly focussed photograph roughly every meter, while pointing the camera straight at the scene. We keep the camera at an approximately constant height and distance from the scene, and keep the zoom level constant between the shots.

To minimize exposure variations, the exposure setting should be set to manual for cameras which support this (ours did not).

Figure 9b shows two possible ways to capture a scene, in a top down plan view. The regular capturing pattern displayed in the left



(a) Acquisition done right. (b) Acquisition done wrong.

Figure 9: Image acquisition done (a) right and (b) wrong.

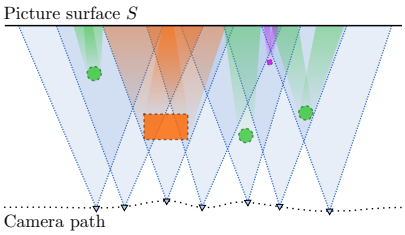


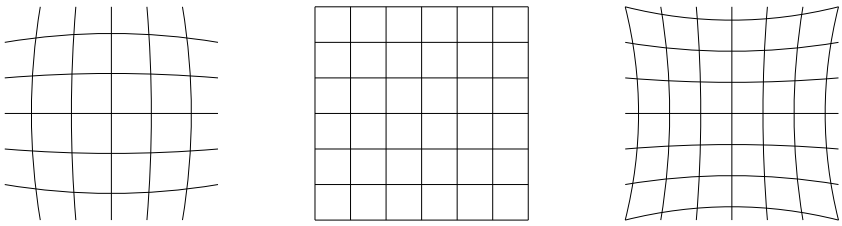
Figure 10: Handling obstructions in the acquisition stage. Small objects like lamp posts (purple circle) are generally easy to look around, even when they are relatively close to the chosen picture surface. Tree stems (green circles) already need more attention. For even bigger objects, like cars (orange rectangle) or tree foliage (not depicted), it may not always be possible to look around them without resorting to very slant views (not depicted).

image follows the rules we have set. This is how we have captured our images. The right image shows a very irregular capturing pattern for which our system is very unlikely to succeed.

If the images for a scene are captured in a completely unorderly fashion and with wildly varying settings, the reconstruction stage is likely to fail. Either no reconstruction is produced at all, or we get an incoherent reconstruction with lots of invalid points and incorrectly inferred camera parameters. Without a proper reconstruction, our system can't produce a panorama.

Additionally, to be able to remove obstructing foreground objects, alternative viewpoints must be available which look around these objects. Depending on the scene, this may require capturing using a lens with a large optical field of view, such as a fisheye lens. See Figure 10 for a visual explanation. This may also be required when shooting scenes where the maximum distance for capturing is physically limited by the environment, which is the case in small streets, for example.

Alternatively, to look around a big object, pictures can be taken at an angle such from which the main surface behind the object is visible. For the results in this paper however, all photographs have been captured using a regular 35mm equivalent camera, and while looking straight at the scene.



(a) Barrel distortion (b) Undistorted (c) Pincussion distortion

Figure 11: Radial distortion.

The aforementioned requirements can be summarized by a small set of simple rules which are easy to follow. The photographer should walk along a path in parallel to the scene and capture a photograph roughly every meter, while facing the scene at a straight angle. This will ensure that we create enough overlap between consecutive images. We also assume that the images are taken with a constant zoom level.

Not a requirement, but still helpful, is setting the exposure and white balance settings of the camera to manual to keep the lighting constant. The cameras we have used did not have this option, which however did not prove too much of a problem.

4.2 PREPROCESSING

While the reconstruction software we are using in the next step can handle a moderate amount of radial distortion, it works better with undistorted, rectangular images. Therefore, we first remove any obvious radial distortion from our source photographs using the *fulla* tool, which comes with the freely available Hugin[11] software. For photographs taken with a fisheye lens this is a required step, but it is also advisable when using rectangular photographs with notable distortion, as such distortion can mess up the reconstruction later on.

In Figure 11, the visual effects of the most common radial distortions are shown. Images exhibiting barrel distortion look like they bulge toward the viewer, while images with pincussion distortion seem to sink into the screen. Such distortions can usually be approximated using a polynomial function.

After this radial distortion correction step, the images are fed into a SfM system to recover the camera parameters and a sparse 3D reconstruction of the scene. We use the Bundler software package developed by Snavely [34, 35], which is described in detail in section 2.4.

The final step of preprocessing involves a simple brightness correction step.

Capturing a large outdoor scene can take a considerable amount of time. During that time, natural phenomena like moving clouds can result in large differences in scene illumination between photographs. By setting the camera to manual exposure, we can prevent the exposure differences between shots to become too large, however, we cannot prevent that some exposure difference will occur. To counteract differently illuminated bands to show up in our final panorama, we will have to use some kind of exposure balancing.

One way to do this is by first recovering the radiometric response curve of each photograph, and then virtually re-expose our source photographs using an exposure-corrected response curve. We have chosen a much simpler approach, which simply assigns a factor k_i to each image I_i such that for each i and j, the color intensities of pixels from features matched between images I_i and I_j are as close to each other as possible: $k_i I_i(p) \approx k_j I_j(p)$ for each matched pixel p. The values for k_i are computed by solving a linear system composed of a set of constraints in a least squares sense.

Each matched keypoint from the reconstructed scene provides 3 such constraints, one for each color channel. We only consider pixels for which neither of the color channels are zero or maxed out, to prevent under- and overexposed pixels from ruining our exposure correction. To prevent that every k_i becomes 0, we further have included a weak prior that every $k_i = 1$.

The linear system is solved by minimizing the sum of squared differences, such that the sum of squared differences is minimal:

$$\min_k S = \sum_{i,j,p \in \mathcal{P}_{i,j}, c \in \{r,g,b\}} \left(k_j \cdot I_j(p)_c - k_i \cdot I_i(p)_c\right)^2 \tag{22}$$

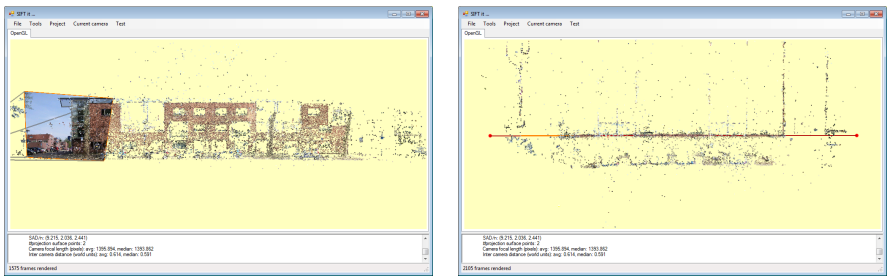
where $\mathcal{P}_{i,j}$ is the set of point matches between images I_i and I_j, and c iterates over the red, green and blue color channels.

4.3 PICTURE SURFACE SELECTION

We load the reconstructed scene into a basic 3D viewer which we have implemented. Upon loading a scene for the first time, the application tries to find a appropriate coordinate system by applying principle component analysis (PCA) to the set of points. The dimension with the greatest variation found by PCA is then used as the new x-axis, and the dimension with the least variation is used as the new y-axis.

Because this approach is not always successful, the user can also define the coordinate system by using the mouse and keyboard to align the point set in an up-right position on screen, and then saving this as the new coordinate system.

After defining the coordinate system, the user can draw a polygonal line in a top-down view of the scene, which will become the picture surface by extruding it in the y-direction. The user can visually position this surface at the correct height using the keyboard.



(a) Frontal view (b) Top-down view

Figure 12: Screenshots of our viewer application, with the GANSSTRAAT data set loaded.

Figure 12 shows two screenshots of our viewer, one frontal and one top-down view.

4.4 COST FUNCTION

After the picture surface has been selected, the user can instruct our viewer to proceed and initiate the panorama creation process.

We first project all images onto the picture surface and save the resulting images as files. After that, we compute the median and MAD images, and save these to image files as well.

Now we can start computing the cost function components. We save every data term of our cost function separately as an image file.

Because we save all our intermediate data as 8-bit TIFF files, every component can only have 256 different values (except for the projected images, which have 8 bit per color channel). While this may seem limited, we have not found that this causes any issues in practice.

Figure 13 shows a visual representation of the data terms, as we have saved them.

4.5 VIEWPOINT SELECTION

When all intermediate data has been computed, we can proceed to compute the labeling for viewpoint selection. For this, we have developed a separate, stand-alone tool, the MRF optimizer.

The MRF optimizer takes a project file as input. In the project file, we specify which images we want to include in the panorama, typically you would use all reconstructed images, but this way, it is also possible to use only a subset. Next to that, we specify the weights for all different terms of the cost function. We have tested several different presets of weights, see Table 3.

The MRF optimizer combines all data terms by applying the weights to the different cost components while loading the images.

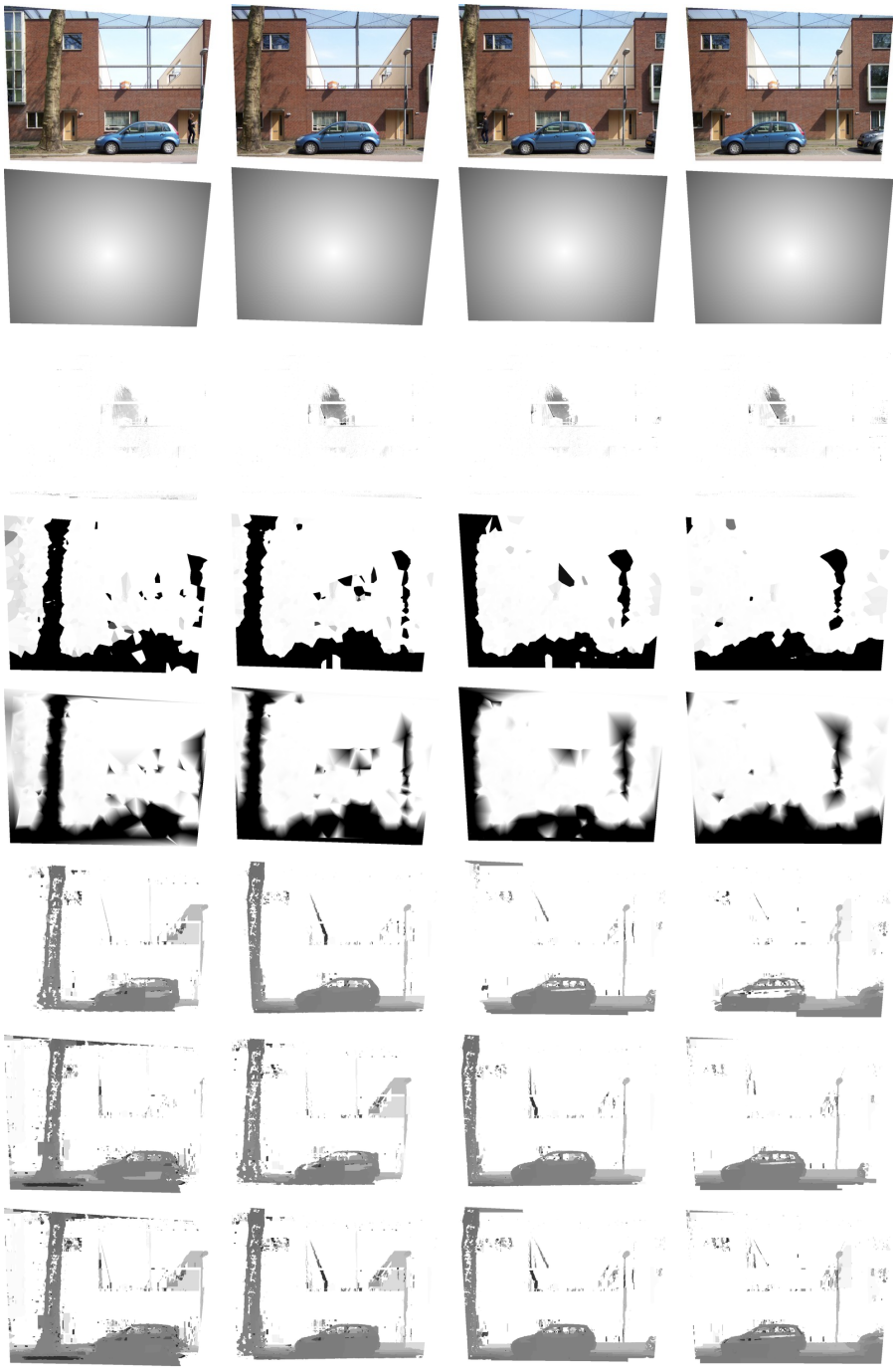


Figure 13: Projected images and data terms, saved as images. From top to bottom: projected images, in-frontness costs, resemble average costs, sparse depth costs (Voronoi, Delaunay), dense depth costs (with image used as left and then right image, and combined).

After that, it starts the MRF optimization and produces its output as two files: the viewpoint selection labeling, and a non-blended panorama.

By default, the MRF optimimizer uses the L_2 distance measure in the computation of the smoothness prior. We have made this distance computation method modular, such that it also allows using the L_1 and L_{inf} distance measure, as well as the sum of squares, which however isn't a metric and does not produce any results. The program comes in two variants: one using integer arithmetic, and one using floating point arithmetic. We have used the floating point version for all the results in this thesis.

4.6 BLENDING

An optional final step is to use Laplacian blending to blend away seams. We use the multiblend tool by Horman [19], which blends all labels at once.

The merged stereo maps in this thesis have also been generated using this tool, with blending disabled such that it uses hard edges instead of smooth ones.

5 RESULTS

To test the performance of our approach, we have run our implementation on various data sets, using various presets for the individual weights of the cost function in Equation 12. In this chapter, we present the results from these experimentations.

To our best knowledge, no established data sets were available, so we have captured a number of scenes ourselves.[1] We have done this in accordance with the rules outlined in Section 4.1.

Table 2 gives an overview of the data sets we have created and used to evaluate our method. A selection of approximately aligned source images is shown for each data set in Figure 14. Each data set has been processed with various settings for the individual weights of our cost function. The weight presets used to obtain the results presented in this paper are shown in Table 3.[2]

In the next sections, we present the results of our experimentations, one data set per section. Besides the constructed panoramas, we also show a distance map, which is computed by applying the labeling from our best result to the individual distance maps of the source photographs computed using our stereo method.

NB: We compare our results to those acquired by using a preset based on the parameters used by our predecessors. While our goals are not exactly the same, the "resemble average" term can be thought of as a poor man's version of foreground object removal. This is also explained in section 3.3.

5.1 KOEKOEKSPLEIN

The scene captured by this data set consists of a housing block, obstructed by 3 trees and a lamp post. For the rightmost tree, no image data is available in the data set which looks around the tree, so it obviously cannot be removed using our approach. All images were taken at an approximately straight angle, from the same height and distance to the principal surface, which we have put at the building facings.

1 Unfortunately the original data sets used by Agarwala et al. could not be obtained due to the data being archived and not readily available.

2 We have actually tested a lot more presets, but these did not improve our results, so we have omitted them from this paper.

Data set	Images		Distance			Section
	t	r	o	p	s	
KOEKOEKSPLEIN	38	37	27	0.8	11	5.1
LODEWIJK NAPOLEONPLANTSOEN	31	31	42	1.4	14	5.2
GANSSTRAAT	75	75	60	0.8	14	5.3
MECKLENBURGLAAN	36	33	50	1.6	16	5.4
ADELAARSTRAAT-2	76	20	90	4.7	12	5.5

Table 2: The data sets used to evaluate our method. Data set name, (t)otal and (r)econstructed number of images, distance between the (o)utermost reconstructed photographs, between each successive (p)air of photographs (both in meters) and approximate distance to the projection (s)urface, and section where the results are presented.

Preset	α	β	γ_v	γ_d	γ_s
AGARWALA	0.4	0.25			
VORONOI-1	0.1		0.2		
VORONOI-2	0.1		0.4		
VORONOI-3	0.1		0.8		
VORONOI-4			1.0		
DELAUNAY-1	0.1			0.2	
DELAUNAY-2	0.1			0.4	
DELAUNAY-3	0.1			0.8	
DELAUNAY-4				1.0	
STEREO-1	0.1				0.2
STEREO-2	0.1				0.4
STEREO-3	0.1				0.8
STEREO-4					1.0

Table 3: The weight presets used by the results in this paper. Weights with a value of 0.0 have been omitted from this table. Note that in their article, Agarwala et al. use a value of 100 for α. This is equivalent to 0.4 in our pipeline because of a slightly different representation of the data.

(a) KOEKOEKSPLEIN

(b) LODEWIJK NAPOLEONPLANTSOEN

(c) GANSSTRAAT

(d) MECKLENBURGLAAN

(e) ADELAARSTRAAT-2

Figure 14: Our data sets, visualized by a selection of the source photographs, appoximately aligned manually by simply translating the images in 2D on the canvas. For the ADELAARSTRAAT-2 data set, we only show the part of the data that was reconstructed (plus a bit more), which is less than a third of the entire set.

In Figure 15 we show our results using a preset based on the weights used by Agarwala et al., and our best results for each of the different depth methods we have implemented.

All our methods (Voronoi, Delaunay and stereo) were able to remove the leftmost and the middle tree. The AGARWALA preset failed to remove the bottom part of the leftmost tree. The STEREO-4 preset was the only one able to remove most of the thin branches otherwise visible in the middle part of the panorama. Compared to the VORONOI-1 and DELAUNAY-1 presets it also succeeded in removing the trash bin and the thin lamp post, which were also removed by the AGARWALA preset.

It however failed to remove the structure in the front in tact whereas the other presets successfully removed it. Inspection of the stereo depth maps confirmed our suspicion that they were not accurate enough in these parts, which is probably caused by the presence of leaves which moved between shots because of the wind, in combination with us failing to keep the camera at a constant enough height.

Note that we had scaled down our (radial distortion corrected) source photographs to 25% for this data set because our machine was severely underpowered at the time. This may also be a cause for inaccurate stereo depth maps. For our other data sets we did not need such a large downscale, resulting in far more accurate disparity maps.

The downscaling was performed using a simple bi-linear interpolation algorithm. More sophisticated interpolation algorithms like bi-cubic and Lanczos resampling have been considered, but on our data sets, we experimentally discovered that the bi-linear approach resulted in more accurate and more complete reconstructions.

It is likely that the full-size images would have generated an even more accurate reconstruction, which we have explored in newer data sets, most notably the Gansstraat data set (see Section 5.3).

5.2 LODEWIJK NAPOLEONPLANTSOEN

Our second data set was captured of a regular block of houses, obstructed by a number of parked cars, as well as two lamp posts and a tree.

For some reason, Bundler was unable to correctly guess the orientation of the coordinate system for this data set, resulting in a heavily rotated reconstruction. Fortunately we can correct for this in our system, and it does not show in the output.

The results computed with our implementation are shown in Figure 16.

For this data set, the best results were achieved using the VORONOI-4 preset.

(a)

(b)

(c)

(d)

Figure 15: KOEKOEKSPLEIN results. Our results using (a) VORONOI-1, (b) DELAUNAY-1, and (c) STEREO-4 presets. (d) Result using the AGARWALA preset. Label borders are highlighted in red.

(e)

(f)

Figure 15: KOEKOEKSPLEIN results (continued). (e) Laplacian blended version and (f) distance map of our STEREO-4 result. Darker pixels are located further away from the user defined picture surface (values adapted for printing purposes).

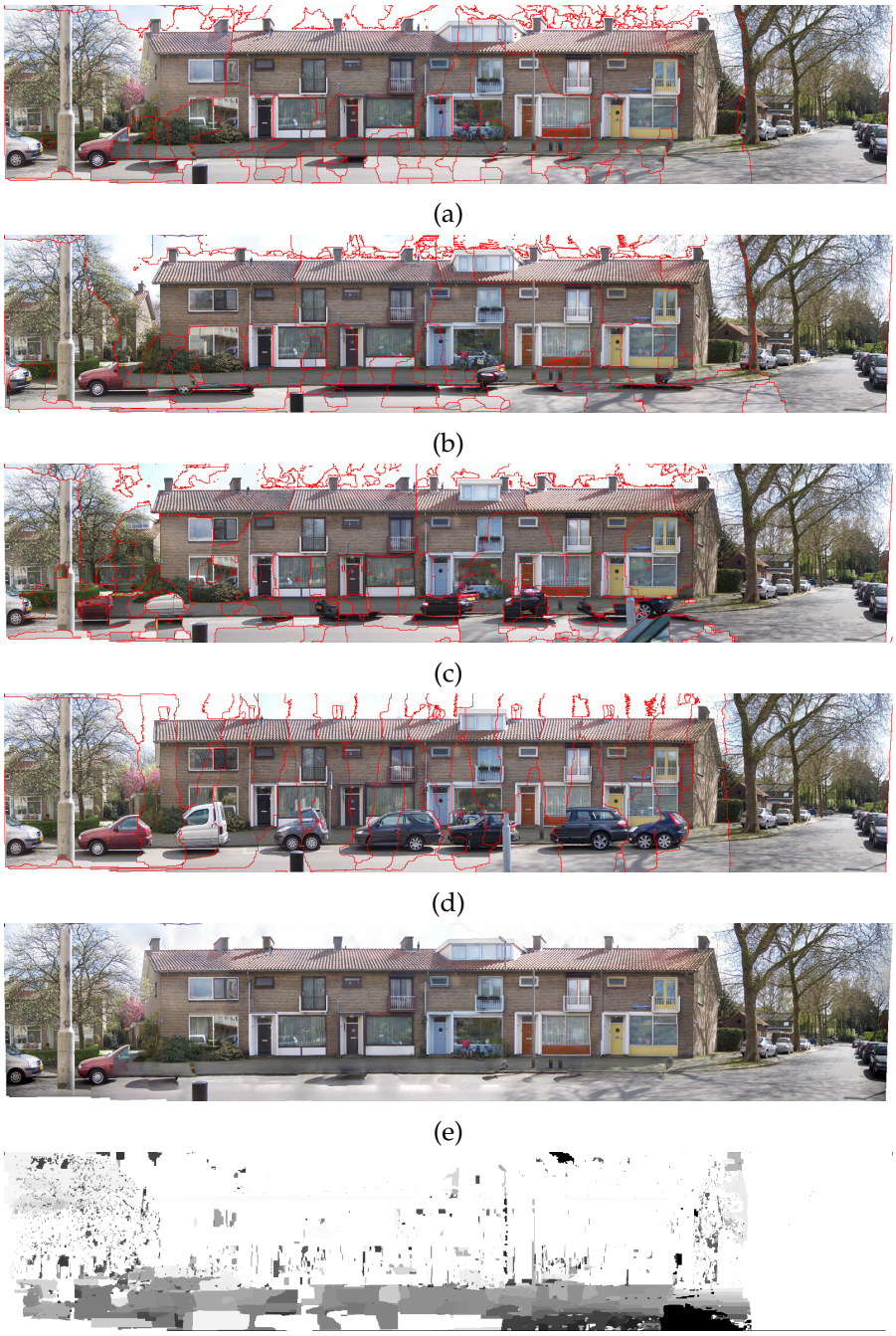
5.3 GANSSTRAAT

We captured a substantially larger set of photographs at Gansstraat, Utrecht. In this set, the building faces are obstructed by some trees, lamp posts and parked cars.

The 75 source photographs taken for this set cover a street length of 60 meters, accounting for an inter-viewpoint distance of about 0.8 meter. Particular care was taken for this set to keep the distance to the picture surface equal, as well as the distance between each consecutive viewpoint and the height of the camera from the ground. Whereas all other data sets were captures with a hand-held camera, for this set we mounted the camera on a bike which we pushed along a line, while capturing a photograph at the exact same intervals. Of all reconstructions, this one was the most precise, with the most reconstructed 3D points.

The STEREO-3 preset performed best on this data set. When looking at the recovered stereo disparities (some of which can be seen on the title page of this thesis) it becomes clear why: they are very precise.

In Figure 17 we present a the results using presets based on Agarwala et al.'s and our approaches.



(a) (b) (c) (d) (e) (f)

Figure 16: LODEWIJK NAPOLEONPLANTSOEN results. Our results using (a) VORONOI-4, (b) DELAUNAY-4, and (c) STEREO-4 presets. (d) Result using the AGARWALA preset. Label borders are highlighted in red. (e) Laplacian blended version and (f) distance map of our VORONOI-4 result. Darker pixels are located further away from the user defined picture surface (values adapted for printing purposes).

(a)

(b)

(c)

(d)

(e)

(f)

Figure 17: GANSSTRAAT results. Our results using (a) VORONOI-3, (b) DELAUNAY-3, and (c) STEREO-3 presets. (d) Result using the AGARWALA preset. Label borders are highlighted in red. (e) Laplacian blended version and (f) distance map of our STEREO-3 result. Darker pixels are located further away from the user defined picture surface (values adapted for printing purposes).

5.4 MECKLENBURGLAAN

This scene depicts another block of houses, obstructed by some parked cars, lamp posts, and trees. The 3 images that were not reconstructed were taken from the leftmost viewpoint while only rotating the camera further to the left, something which our SfM cannot deal with.

What's special about this scene is that the trees are very near to the surface. As a result, it is impossible to remove them entirely with our technique. Nevertheless, only a small fraction of them is left in our best result, which was achieved using the DELAUNAY-2 preset.

The street in this scene is pretty wide, allowing a fair distance between the camera viewpoint and the buildings we were interested in. As a result, the overlap between consecutive images is fairly large: at over 90%, each pixel of the picture surface is seen by about 12 cameras. Not surprisingly, the 3D reconstruction for this scene was relatively dense.

This also explains why the best result could be achieved with the DELAUNAY-2 preset, with VORONOI-2 as a very close second. Both of these presets are based on the sparse depth map which use the sparse reconstruction as input. And both succeed in removing the cars and the stems of the trees almost entirely, with only their shadows left as evidence. To be fair, the results of our STEREO-2 preset did a slightly better job of removing the tree foliage, which is best seen in the 3rd tree from the right.

When we compare our results to the AGARWALA preset, it becomes clear that their method really was not targetted at removing foreground objects. They actually did not remove any car at all, although none of them has been kept in one piece. The same is true for the trees.

In Figure 18 we show our results on this data set.

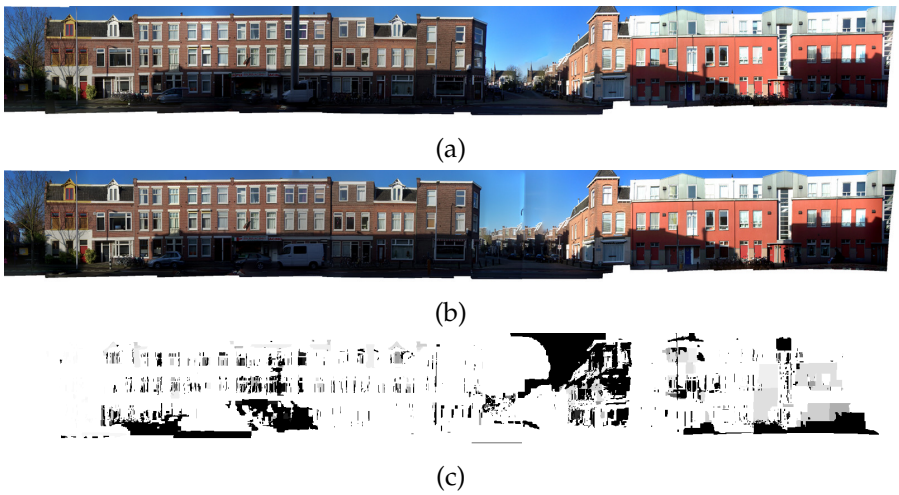
5.5 ADELAARSTRAAT-2

In Figure 19 we present some of the results from the ADELAARSTRAAT-2 data set.

Basically our method failed on this data set, regardless of the preset. Therefore, we only present two results: one using our VORONOI-4 preset, which has failed the least, and one using the AGARWALA preset. Their performance is more or less equivalent in this case. Our VORONOI-4 result removed more of the white van and the other cars, but did manage to include a lamp post in the result which was filtered out by the AGARWALA preset. The other results were approximately in between those results, but none of them succeeded in removing all foreground objects.

(a) (b) (c) (d) (e) (f)

Figure 18: MECKLENBURGLAAN results. Our results using (a) VORONOI-2, (b) DELAUNAY-2, and (c) STEREO-2 presets. (d) Result using the AGARWALA preset. Label borders are highlighted in red. (e) Laplacian blended version and (f) distance map of our DELAUNAY-2 result. Darker pixels are located further away from the user defined picture surface (values adapted for printing purposes).



(a)

(b)

(c)

Figure 19: ADELAARSTRAAT-2 results. Laplacian blended version of the result using (a) our VORONOI-4 and (b) the AGARWALA preset. (c) Distance map of our VORONOI-4 result. Darker pixels are located further away from the user defined picture surface (values adapted for printing purposes).

As to why this data set proved so difficult, let us have a look at the source photographs. As can be seen in Table 3, for this data set the photographs were captured with much more distance between each consecutive camera than for the other data sets. Additionally, the distance of the camera viewpoints to the picture surface is relatively small. Together, this makes for an overlap between neighboring images of only about 70%, which results in each pixel on the picture surface being seen by at most 4 viewpoints in the optimal case.

The structure on the right of the image is repeated several more times in reality, but SfM failed to identify them as separate buildings, so we could only use a very small fraction of the actual data set. All in all, SfM only reconstructed 20 images out of 76 successfully. Another 18 cameras were reconstructed incorrectly, with bogus camera parameters and reconstructed points. We have disabled those images to obtain at least some kind of valid result.

Another thing is that the street is quite narrow, forcing the cars to be parked very near to the buildings. This makes it hard to look around them, which is why we could not remove any of them entirely.

And because of the large baseline, stereo was not effective.

6 DISCUSSION

From the results presented in the previous chapter it is clear that stereo information can be used to remove foreground objects in multi-viewpoint panoramas.

Not every object can be removed using our approach. Objects which are too big or are located too close to the picture surface usually form a problem, because they can't be looked around, at least not without taking the photographs at a strong angle. The trees in our MECKLENBURG data set for example almost touch the buildings, and are therefore only removed partly. Inpainting techniques, which are part of the original pipeline by Agarwala et al., may help overcome this issue, but we haven't implemented that.

6.1 STRUCTURE-FROM-MOTION

Many different issues can arise with SfM, unfortunately often beyond our control. We will mention some of those problems and inconveniences here.

Because of those issues, we actually had to capture a lot more data sets than the ones included in this thesis. Fortunately, the data sets that did make it still show that our method works.

To be able to create a relatively dense reconstruction, SfM needs to be fed with as many point matches as possible. Unfortunately, certain kinds of scenes will not permit this. For example, SIFT will not find many keypoints if the scene contains large textureless regions. Since the point matches are constructed from these keypoints, this will result in less point matches.

Reflective surfaces are a problem, because point matches from such surfaces do not depict the true locations of scene geometry, but rather of their reflections. This does not only apply to windows, but many times also to cars, due to their shiny finish.

Another problem for SfM is the presence of repetitive features, such as patches of equally spaced windows. This is to be expected because the system cannot usually discern between objects which look exactly similar. Even when using techniques which identify such repetitions (which the implementation we have used does not include), this will go wrong when the repetition goes on for too long, i.e. longer than what can be captured in a single photograph. We found that when these repetitive structures make up a big part of the image, the presence of other features, which may allow a human to identify the cor-

rect matches manually, may not be enough to get a good reconstruction.

SfM expects the scene to be rigid, which in reality is not always the case. During capturing, people may move through the scene, clouds may move (also causing lighting changes), etc. When enough input photographs are available, this need not be a problem, and we have not had any issues with it.

SfM methods usually also require that the point matches are spread in 3 dimensions to at least some degree. When all point matches are found on a single flat surface, homographies are ill-defined and the reconstruction will fail.

6.2 DEPTH-BASED TERMS

We have found that in our implementation, neither of the depth-based depth terms produces the best result under all circumstances.

This can be explained at least in part by the quality of the sparse 3D reconstruction. When the reconstruction is denser, the depth maps created by our Voronoi and Delaunay methods will be more accurate. In this case, the stereo based depth map may produce better results. Also, SfM is better at handling occlusions, because it handles more viewpoints (essentially, it is a sparse version of multi-view stereo).

On the other hand, for our stereo based depth map to produce high quality results, it is essential that the camera was moved in a line strictly parallel to the scene and to the x-axis of the picture surface during capturing. If too little attention is paid to this rule, our approximation for epipolar geometry will be incorrect, resulting in objects displayed at a different vertical position in the left and right image. The approach is then likely to fail on objects further away from the picture surface.

Our stereo method also relies on the order in which the images were taken, which has severe implications if the images are somehow shuffled. Because it assumes that every camera C_i is to the right of C_{i-1}, our stereo method also cannot currently handle multi-row panoramas. SfM does not have these limitations and can handle input of any ordering.

Just like SfM, stereo disparity algorithms are dependent on the presence of texture in the scene. In the absence of texture, the algorithm cannot find corresponding points in the left and right input images.

6.3 OTHER LIMITATIONS

While working on the thesis and the applications we have been developing, we have found that too much data can be a problem, at least on our system. When this project was started, our workstation was a lap-

top with 4GB of RAM, partly used as video memory. When loading 75 images at 5MP each (which is at least 15MB per image, but more likely 20MB if it is stored as RGBA), already over 1.1GB of memory is used by the images alone. For various computations, multiple copies of at least some of the images are required to be present in memory. Add to that the scene reconstruction, some overhead, the application itself, and other running applications, and the memory will become a bottleneck.

After upgrading to 8GB of RAM, some of the problems disappeared, but others persisted. For instance, we have tried to render our GANSSTRAAT data set at full resolution. The reconstructed scene was pretty dense, with over 900k points. Our application could load this data set, but crashed immediately after upon trying to do anything at all. Therefore, all of our data sets have been run through the pipeline at half resolution.

7 CONCLUSION

7.1 CONCLUSION

We have implemented a pipeline for the construction of multi-viewpoint panoramas, based on the work of Agarwala et al. [3]. We have adapted this pipeline for the purpose of foreground object removal. This was done by adding a new, depth-based data term to the cost function we optimize to find a labeling for viewpoint selection. The depth maps for this cost function are computed from sparse depth from the 3D reconstruction computed during the preprocessing stage, as well as using a stereo disparity algorithm on the projected photographs.

We have successfully demonstrated that our approach works for real-world data. works improve foreground object removal in the application of multi-viewpoint panoramas.

7.2 FUTURE WORK

The parameter preset which we have tried do not cover every possible combination. Testing this is practically impossible. Some parameters are also dependent on the physical size of the input (scene geometry) and output settings (panorama resolution). For instance, the number of disparity levels tried in our stereo algorithm is currently hard-coded, while it is in fact dependent on the physical size of a pixel. The same holds for the factor which we use internally to cap the Voronoi and Delaunay depth maps.

It may be possible to get rid of at least some of these dependencies by making them size-invariant.

We have already pointed out that the stereo disparity algorithm which we have used is not state-of-the-art. Better algorithms are available for 2-view stereo, some of which can compute depth with sub-pixel precision and thus better depth resolution. Even better would be to use algorithms capable of doing multi-view stereo, like Zhang et al. [42]. Losing the dependence on image ordering, we would probably also be able to handle multi-row input.

Since we began working on this thesis, frameworks have been developed which use GPU processing for several of the computations we use throughout this thesis. For instance, VisualSFM [39] is a framework for SfM, which can compute keypoints and bundle adjustment on the GPU, while displaying the results in real-time. It can also run

repetition analysis [40, 41], which may help overcome the issues with repetitive structures we ran into.

Another thing which our approach cannot handle is large jumps in depth, i.e. our picture surface cannot contain depth incontinuities. As a result, our approach is currently limited to scenes where the objects approximately form a line, as seen from a top-down view.

Our MRF optimization step could be sped up significantly by merging labels and their associated projected and cost images for which the projected images do not share any pixels. In our data sets the overlap between neighboring images varies from about 70% to about 90%, which means that image I_i usually does not overlap image I_j when $j > i + 10$. This can be computed automatically by looking at the actual pixels (or, more simple, the bounding boxes).

Obviously, we can expect more speedup and improved output by implementing the hierarchical MRF optimization approach, the gradient domain blending, and the interactive refinement step from Agarwala et al. [3].

Finally, we should mention that our implementation is not very well optimized. Some values are cached, while others are computed over and over again. For speedup, more caching should be used. This may be improved even further by switching from C# to native C++ or C, which will remove a few layers of abstraction in several parts of our application (OpenGL and OpenCV for example).

The panorama-city is a 'theoretical' (that is, visual) simulacrum, in short a picture, whose condition of possibility is an oblivion and a misunderstanding of practices.

— Michel de Certeau, *The Practice of Everyday Life* [9]

ACKNOWLEDGMENTS

As my initial supervisor, Twan Maintz has played a vital role in finding a subject for my master thesis research project. I had found this great article and had a lot of crazy ideas on how to modify and improve the workflow presented in the article. Twan and I have had numerous discussions on various aspects of panoramic photography, cameras, images and 3D. Some ideas were fruitful but found a bit too obvious by Twan, even though I wasn't too sure about the straightforwardness of some of those problems. Finally we settled on the current subject of this thesis, and I started working.

Under Twan's supervision, I reimplemented a big part of the system described by the authors of the article my research is based upon, plus a big part of the proposed extensions. Unfortunately, due to several reasons, it took me a lot longer to get things working the way I wanted. This dimmed my enthousiasm, and over time resulted in the project pretty much grinding to a halt.

I thought about giving up, but this would mean all my hard work would go to waste. So after a period of little activity, I pulled myself together and got back to work. Because Twan's job responsibilities had changed in the meantime, he could no longer supervise me. Luckily, I found a new supervisor in Robby Tan, who helped me get back on track, and finally, finish my thesis. Without his extensive experience in the field, positive attitude and determination, I would never have made it, so I owe him a debt of gratitude.

During the course of the project, there have been a lot of people who stood by my side whenever I needed support. I cannot express how much I love you for that. Thank you all so much! I especially want to thank my parents, my brother and sisters for their patience and unconditional trust in me. Even when I wanted to give up sometimes, they were there for me to motivate me and cheer me up enough to keep me going. Some friends have made fun of me for being the typical showcase of an "eternal student". It feels good to prove them wrong in the end!

Last but not least, my loving girlfriend never stopped supporting me, so she definitely deserves a lot of credit. She said she would kick my ass if I gave up. I guess I am safe now.

What a relief!

BIBLIOGRAPHY

[1] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven M. Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael F. Cohen. Interactive digital photomontage. *ACM Trans. Graph.*, 23(3):294–302, 2004. doi: 10.1145/1015706.1015718. URL http://research.microsoft.com/apps/pubs/default.aspx?id=80404.

[2] Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael F. Cohen, Brian Curless, David Salesin, and Richard Szeliski. Panoramic video textures. *ACM Trans. Graph.*, 24(3):821–827, 2005. doi: 10.1145/1073204.1073268. URL http://research.microsoft.com/apps/pubs/default.aspx?id=74506.

[3] Aseem Agarwala, Maneesh Agrawala, Michael F. Cohen, David Salesin, and Richard Szeliski. Photographing long scenes with multi-viewpoint panoramas. *ACM Trans. Graph.*, 25(3):853–861, 2006. doi: 10.1145/1141911.1141966. URL http://research.microsoft.com/apps/pubs/default.aspx?id=74507.

[4] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit S. Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *IEEE Computer*, 43(6):32–38, 2010. doi: 10.1109/MC.2010.170.

[5] Jaakko Astola, Petri Haavistom, and Yrjö Neuvo. Vector median filters. In *Proceedings of the IEEE*, volume 78 of 4, pages 678–689, 1990. doi: 10.1109/5.54807.

[6] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *ICCV*, pages 377–384, 1999.

[7] Jonathan Broere. Urban scene reconstruction using image sequences. Master's thesis, Utrecht University, October 1 2007.

[8] Matthew Brown and David G. Lowe. Recognising panoramas. In *ICCV*, pages 1218–1227. IEEE Computer Society, 2003. ISBN 0-7695-1950-4.

[9] Michel De Certeau. *The Practice of Everyday Life*. University of California Press, 1984. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0520236998.

[10] Chia-Yen Chen and Reinhard Klette. Image stitching - comparisons and new techniques. In Franc Solina and Ales Leonardis, editors, *CAIP*, volume 1689 of *Lecture Notes in Computer Science*, pages 615–622. Springer, 1999. ISBN 3-540-66366-5.

[11] Pablo d'Angelo. Hugin - panorama photo stitcher, 2003 . URL http://hugin.sourceforge.net/.

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6): 381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692.

[13] Christian Früh and Avideh Zakhor. An automated method for large-scale, ground-based city model acquisition. *International Journal of Computer Vision*, 60(1):5–24, 2004.

[14] Christian Früh, Russell Sammon, and Avideh Zakhor. Automated texture mapping of 3d city models with oblique aerial imagery. In *3DPVT*, pages 396–403. IEEE Computer Society, 2004. ISBN 0-7695-2223-8.

[15] Google. Minnaertgebouw, leuvenlaan, utrecht, netherlands – google street view, 2009. URL https://maps.google.com/?cbll=52.086214,5.16689&layer=c&cbp=13,-7.276563998161458,,0,-3.036676314193258. [Online; accessed 12-July-2012].

[16] Alison J. Gray, Jim Kay, and D. M. Titterington. On the estimation of noisy binary markov random fields. *Pattern Recognition*, 25(7): 749–768, 1992.

[17] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):580–593, 1997.

[18] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[19] David Horman. multiblend. URL http://horman.net/multiblend/.

[20] Ross Kindermann and J. Laurie Snell. *Markov Random Fields and Their Applications*. AMS, 1980. URL http://www.ams.org/online_bks/conm1/.

[21] Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions via graph cuts. In *ICCV*, pages 508–515, 2001.

[22] Johannes Kopf, Billy Chen, Richard Szeliski, and Michael F. Cohen. Street slide: browsing street level imagery. *ACM Trans. Graph.*, 29(4):96:1–96:8, 2010. doi: 10.1145/1778765.1778833. URL http://research.microsoft.com/apps/pubs/default.aspx?id=121744.

[23] Vivek Kwatra, Arno Schödl, Irfan A. Essa, Greg Turk, and Aaron F. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003.

[24] Vivek Kwatra, Irfan A. Essa, Aaron F. Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24(3):795–802, 2005.

[25] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[26] John Mallon and Paul F. Whelan. Projective rectification from the fundamental matrix. *Image Vision Comput.*, 23(7):643–650, 2005.

[27] Murali M. Menon. An efficient mrf image-restoration technique using deterministic scale-based optimization. *Lincoln Laboratory Journal*, 6(1):147–160, 1993. doi: 10.1.1.186.2586.

[28] Microsoft. Streetside, the true-to-life experience for explorers everywhere. URL http://www.microsoft.com/maps/streetside.aspx. [Online; accessed 13-July-2012].

[29] Tijana Ružić, Aleksandra Pižurica, and Wilfried Philips. Markov random field based image inpainting with context-aware label selection. Accepted by the 2012 IEEE International Conference on Image Processing, October 2012.

[30] Daniel Scharstein and Anna Blasiak. Middlebury stereo evaluation - version 2. URL http://vision.middlebury.edu/stereo/eval/. [Online; accessed 9-August-2012].

[31] Daniel Scharstein and Richard Szeliski. Middlebury stereo vision page. URL http://vision.middlebury.edu/stereo/. [Online; accessed 13-July-2012].

[32] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[33] Thomas K. Sharpless, Bruno Postle, and Daniel M. Germán. Pannini: A new projection for renderingwide angle perspective images. In Pauline Jepp and Oliver Deussen, editors, *Computational Aesthetics*, pages 9–16. Eurographics Association, 2010. ISBN 978-3-905674-24-8.

[34] Noah Snavely. Bundler: Structure from motion (sfm) for unordered image collections, April 2010. URL http://phototour.cs.washington.edu/bundler/. [Online; accessed 10-January-2012].

[35] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, 2006. doi: 10.1145/1141911.1141964. URL http://research.microsoft.com/apps/pubs/default.aspx?id=75618.

[36] Noah Snavely, Rahul Garg, Steven M. Seitz, and Richard Szeliski. Finding paths through the world's photos. *ACM Trans. Graph.*, 27(3):15:1–15:11, August 2008. doi: 10.1145/1360612.1360614. URL http://research.microsoft.com/apps/pubs/default.aspx?id=75620.

[37] Richard Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–104, January 2006. ISSN 1572-2740. doi: 10.1561/0600000009. URL http://research.microsoft.com/apps/pubs/default.aspx?id=75695.

[38] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall F. Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(6):1068–1080, 2008.

[39] Changchang Wu. VisualSFM: A Visual Structure from Motion System, 2011.

[40] Changchang Wu, Jan-Michael Frahm, and Marc Pollefeys. Detecting large repetitive structures with salient boundaries. In *Proceedings of the 11th European conference on Computer vision: Part II*, ECCV'10, pages 142–155, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15551-0, 978-3-642-15551-2. URL http://dl.acm.org/citation.cfm?id=1888028.1888040.

[41] Changchang Wu, Jan-Michael Frahm, and Marc Pollefeys. Repetition-based dense single-view reconstruction. In *CVPR*, pages 3113–3120. IEEE, 2011. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2011.html#WuFP11.

[42] Guofeng Zhang, Jiaya Jia, Tien-Tsin Wong, and Hujun Bao. Consistent depth maps recovery from a video sequence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(6):974–988, 2009. URL http://www.cad.zju.edu.cn/home/gfzhang/projects/videodepth/.

COLOPHON

This document was typeset in LaTeX using the typographical look-and-feel `classicthesis` developed by André Miede. Graphics were created and/or edited using a combination of Inkscape, TikZ, and the GIMP. Editing was done using Texlipse, TeXworks, and Notepad++.

Final Version as of August 22, 2012 (`Master Thesis` version 1.0).