Immersive Automotive Stereo Vision

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

M.Sc. Jonas Haeling
aus Wittingen

Tübingen
2020

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	19.02.2021
Stellvertretender Dekan:	Prof. Dr. József Fortágh
1. Berichterstatter:	Prof. Dr. Andreas Schilling
2. Berichterstatter:	Prof. Dr.-Ing. Hendrik P.A. Lensch

In memory of Mirco.

Abstract

Recently, the first in-car augmented reality (AR) system has been introduced to the market. It features various virtual 3D objects drawn on top of a 2D live video feed, which is displayed on a central display inside the vehicle. Our goal with this thesis is to develop an approach that allows to not only augment a 2D video, but to reconstruct a 3D scene of the surrounding driving environment of the vehicle. This opens up various possibilities including the display of this 3D scan on a head-mounted display (HMD) as part of a Mixed Reality (MR) application, which requires a convincing and immersive visualization of the surroundings with high rendering speed. To accomplish this task, we limit ourselves to the use of a single front-mounted stereo camera on a vehicle and fuse stereo measurements temporally.

First, we analyze the effects of temporal stereo fusion thoroughly. We estimate the theoretically achievable accuracy and highlight limitations of temporal fusion and our assumptions. We also derive a 1D extended information filter and a 3D extended Kalman filter to fuse measurements temporally, which substantially improves the depth error in our simulations. We integrate these results in a novel dense 3D reconstruction framework, which models each point as a probabilistic filter. Projecting 3D points to the newest image allows us to fuse measurements temporally after a clustering stage, which also gives us the ability to handle multiple depth layers per pixel.

The 3D reconstruction framework is point-based, but it also has a mesh-based extension. For that, we leverage a novel depth image triangulation method to render the scene on the GPU using only RGB and depth images as input. We can exploit the nature of urban scenery and the vehicle movement by first identifying and then rendering pixels of the previous stereo camera frame that are no longer seen in the current frame. These pixels at the previous image border form a tube over multiple frames, which we call a tube mesh, and have the highest possible observable resolution. We are also able to offload intensive filter propagation computations completely to the GPU. Furthermore, we demonstrate a way to create a dense, dynamic virtual sky background from the same camera to accompany our reconstructed 3D scene.

We evaluate our method against other approaches in an extensive benchmark on the popular KITTI visual odometry dataset and on the synthetic SYNTHIA dataset. Besides stereo error metrics in image space, we also compare how the approaches perform regarding the available depth structure in the reference depth image and in their ability to predict the appearance of the scene from different viewing angles on SYNTHIA. Our method shows significant improvements in terms of disparity and view prediction errors. We also achieve such a high rendering speed that we can fulfill the framerate requirements of modern HMDs. Finally, we highlight challenges in the evaluation, perform ablation studies of our framework and conclude with a qualitative showcase on different datasets including the discussion of failure cases.

Kurzfassung

Kürzlich wurde das erste In-Car Augmented Reality (AR) System eingeführt. Das System beinhaltet das Rendern von verschiedenen 3D Objekten auf einem Live-Video, welches auf einem Zentraldisplay in der Mittelkonsole des Fahrzeuges angezeigt wird. Ziel dieser Arbeit ist es ein System zu entwickeln, welches nicht nur 2D-Videos augmentieren kann, sondern eine 3D-Rekonstruktion der aktuellen Fahrzeugumgebung erstellen kann. Dies ermöglicht eine Vielzahl von verschiedenen Anwendungen, u.a. die Anzeige dieses 3D-Scans auf einem Head-mounted Display (HMD) als Teil einer Mixed Reality (MR) Anwendung. Eine MR-Anwendung bedarf einer überzeugenden und immersiven Darstellung der Umgebung mit einer hohen Renderfrequenz. Wir beschränken uns auf eine einzelne Front-Stereokamera, welche vorne am Auto verbaut oder montiert ist, um diese Aufgabe zu bewältigen. Hierzu fusionieren wir die Stereomessungen temporär.

Zuerst analysieren wir von Grund auf die Effekte der temporalen Stereofusion. Wir schätzen die erreichbare Genauigkeit ab und zeigen Einschränkungen der temporalen Fusion und unseren Annahmen auf. Wir leiten außerdem ein 1D Extended Information Filter und ein 3D Extended Kalman Filter her, um Stereomessungen temporär zu vereinen. Die Filter verbesserten den Tiefenfehler in Simulationen wesentlich. Die Ergebnisse der Analyse integrieren wir in ein neuartiges 3D-Rekonstruktions-Framework, bei dem jeder Punkt mit einem Filter modelliert wird. Das sog. “Warping” von Pixeln von einem Bild zu einem anderen Bild ermöglicht die temporäre Fusion von Messungen nach einem Clustering-Schritt, welcher uns erlaubt verschiedene Tiefenebenen pro Pixel gesondert zu betrachten.

Das Framework funktioniert als punkt-basierte Rekonstruktion oder alternativ als mesh-basierte Erweiterung. Hierfür triangulieren wir Tiefenbilder, um die 3D-Szene nur mit RGB- und Tiefenbildern als Input auf der GPU zu rendern. Wir können die Eigenschaften von urbanen Szenen und der Kamerabewegung ausnutzen, um Pixel zu identifizieren und zu rendern, welche nicht mehr in zukünftigen Frames beobachtet werden. Das ermöglicht uns diesen Teil der Szene in der größten beobachteten Auflösung zu rekonstruieren. Solche Randpixel formen einen Schlauch (“Tube”) über mehrere Frames, weshalb wir dieses Mesh als Tube Mesh bezeichnen. Unser Framework erlaubt es uns auch die rechenintensiven Filter-Propagationen komplett auf die GPU auszulagern. Des Weiteren demonstrieren wir ein Verfahren, um einen vollen, dynamischen, virtuellen Himmel mithilfe der gleichen Kamera zu erstellen, welcher ergänzend zu der 3D-Szenenrekonstruktion als Hintergrund gezeigt werden kann.

Wir evaluieren unsere Methoden gegen andere Verfahren in einem umfangreichen Benchmark auf dem populären “KITTI Visual Odometry”-Datensatz und dem synthethischen SYNTHIA-Datensatz. Neben Stereofehlern im Bild vergleichen wir auch die Performanz der Verfahren für die Rekonstruktion von bestimmten Strukturen in den Referenz-Tiefenbildern, sowie ihre Fähigkeit die Erscheinung der 3D-Szene

aus unterschiedlichen Blickwinkeln vorherzusagen auf dem SYNTHIA-Datensatz. Unsere Methode zeigt signifikante Verbesserungen des Disparitätsfehlers sowie des Bildfehlers aus unterschiedlichen Blickwinkeln. Außerdem erzielen wir eine so hohe Rendergeschwindigkeit, dass die Anforderung der Bildwiederholrate von modernen HMDs erfüllt wird. Zum Schluss zeigen wir Herausforderungen in der Evaluation auf, untersuchen die Auswirkungen des Weglassens einzelner Komponenten unseres Frameworks und schließen mit einer qualitativen Demonstration von unterschiedlichen Datensätzen ab, inklusive der Diskussion von Fehlerfällen.

Acknowledgements

This work was carried out during my time at the Group Research & MBC Development of Mercedes-Benz AG, Sindelfingen in Germany.

First, I would like to thank Dr.-Ing. Marc Necker and Prof. Dr. Andreas Schilling for providing this great opportunity to me and for their helpful advice and support over multiple years. Thanks also goes to Prof. Dr.-Ing. P.A. Hendrik Lensch, Dr. Benjamin Resch, Sascha Meyen and Manuel Lange from the Eberhard-Karls-Universität Tübingen for the very fruitful discussions.

I would also like to express my thanks to my colleagues at the Mercedes-Benz AG: Dr. Christian Grünler, Dr. Janis Werner, Dr. Christoph Will, Dr. Stefan Gehrig, Dr. Stephan Schmid, Dr. Johannes Rabe and Xiang Gao. Your feedback and advice was invaluable. I also have to thank all of the members of the Augmented Reality team, who supported me, especially Tobias Staib and Lennart Planz for their excellent work.

Furthermore, I would like to express my thanks to my brother Christian and also to Sebastian Starke, both of whom motivated me greatly to pursue my intellectual interests. Lastly, I would like to thank Felicia for her love, care and patience while writing this thesis.

Contents

Chapter 1

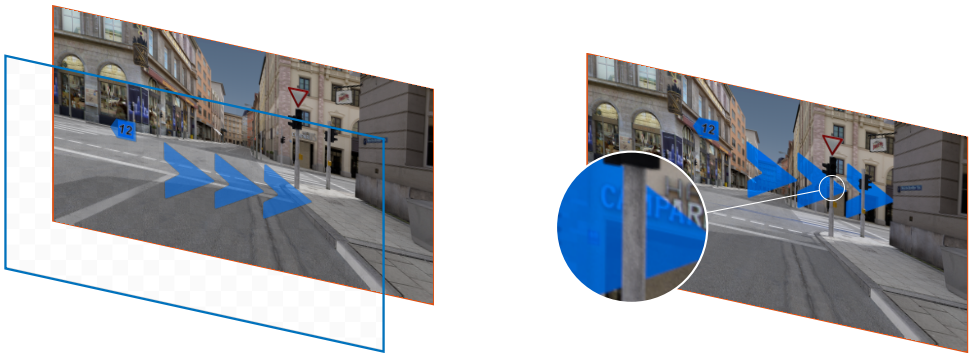
Introduction

1.1 Goal of this Thesis

Automotive navigation systems, which show directions on a digital map, are common accessories in modern cars. The digital map can even be a rudimentary 3D view of landmark buildings with mapped satellite textures. Still, this abstract map representation has to be interpreted by the driver and mapped to the real world. Recently, an Augmented Reality system for navigation on an in-vehicle display using a front camera (see [Grü13]) was developed by the car manufacturer Mercedes-Benz as a first-to-market feature.

FIGURE 1.1: Mercedes-Benz' AR system [Dai19]. Traditional map is on the left, while the augmented video application shows a turn right maneuver with the associated street name in the video.

Figure 1.1 shows a close up of this system. The goal of this system is to provide a more intuitive presentation of navigation and other information to the vehicle driver and passengers. Navigation arrows, street names, house numbers and points of interests such as the target location are displayed in 3D where they would belong in the real world. This requires a fast and precise pose estimate of the vehicle to keep the



(a) Image with rendered 3D objects as an overlay. (b) Image of a 3D scene with 3D objects.

FIGURE 1.2: Comparison between 2D and possible 3D AR extension.

rendered virtual 3D objects aligned to the video feed, *i.e.* by estimating the relative coordinate system of the camera to the world. For example, a small bump in the road will cause a change of pitch of the vehicle, which needs to be reacted to immediately in AR to keep the virtual 3D objects stationary in the live video.

Still, there are two shortcomings of this system. First, the driver has to take their eyes off the real road to look at the central display. This is the same for conventional navigation systems, but we still see the road scene ahead in the video with such an AR system. Second, the virtual content is only an overlay, which is always drawn on top of the video. A possible 3D extension, *i.e.* incorporating knowledge about the scene geometry, allows for an immersive display of virtual content as illustrated in Figure 1.2. The left image only uses a virtual overlay, while the right image is a rendering of a 3D scene.

Depth and normal data for each video pixel could open up a wide range of possible effects: Proper occlusion of virtual objects behind real objects would be possible as shown in the magnifying glass of Figure 1.2b. Furthermore, normal information, which describes the perpendicular direction of the tangent on a geometrical object, would allow an increased immersion by relighting the scene (*e.g.* glowing virtual objects), ambient occlusion, reflectancy, shadows and virtual objects pinned to real geometry. Note that shadows of virtual objects exist on the road in Figure 1.2b. Even physical simulation effects would become possible, *e.g.* with a connected 3D road surface. One could let virtual objects interact with the real world via collision detection, for example.

During an autonomous drive, AR and also Virtual Reality (VR) may provide new and interesting ways to interact with the car and the environment. As the driver is no longer burdened with the act of driving the car itself, a variety of entertainment application possibilities opens up. Particularly, HMDs like the Oculus Quest [Fac19a], which allows an untethered immersive VR experience with tracked hand-controllers, can be considered. For example, potential gaming and entertainment use cases can include a VR roller coaster ride along a navigation route, in which the user rides through a fictitious landscape. The landscape could react dynamically to the vehicle

movement and the current progress along the route. In-car actuators such as the air conditioning or seat massage functions could enhance the immersion even further by supporting dynamic events and effects in VR.

Furthermore, the aforementioned scanned 3D geometry of a scene can be combined with a VR HMD for Mixed Reality. It would be similar to a stereo pass-through system which 3D reconstructs the current view, but with a vehicle-mounted camera (instead of being directly mounted on the VR HMD). This 3D scan would allow a free viewpoint interpolation in the car. Such a system may also extend previous surround view camera systems, which use panoramic stitching methods to provide a 360° video. Generally, these systems rely on monocular cameras without depth estimation, which means that positional movement of the observer cannot be accounted for, but only rotational movement. The front camera for the aforementioned AR application mainly sees the driving scene ahead and not the car itself. This means that the vehicle is not included in a 3D rendering of the scene, which enables augmenting a different vehicle cockpit, *e.g.* a helicopter cockpit or none at all.

One often overlooked but very relevant aspect in autonomous driving is motion sickness according to Diels and Bos [DB16], which we will detail in the following. With increasing levels of automation, *i.e.* relieving the driver of more and more driving tasks to a fully autonomous ride, there is an increased risk of motion sickness. Motion sickness is not a disease, but more so a natural response to an unnatural environment and includes a wide range of symptoms including nausea, vomiting or the feeling of dizziness. Especially non-driving tasks or even a rearward faced seating arrangement may be critical. This may be caused by the passenger's inability to predict the motion trajectory of the vehicle or the occlusion of the real world motion in the rear-seated case. In contrast, drivers rarely get sick.

There is no accepted general theory on motion sickness. It is hypothesized that the central nervous system integrates both visual and vestibular signals (inner ear, a biological accelerometer) and that motion sickness may be caused by sensory conflicts. For example, reading a large newspaper while seated in a vehicle blocks the visual perception of motion, but the felt accelerations communicate that there is indeed movement. Also, some passengers may get sick looking at 2D screens in a vehicle for the same reason. Stronger physical forces like hard accelerations, braking and cornering will increase motion sickness.

In the human retina, most visual motion perceiving cells lie on the outside of the fovea (cf. Hersh [Her08]). A potential counter-measure visualization would thus need a large field of view (FoV) to display the moving vehicle dynamics in some way, *e.g.* on an HMD. However, this needs to be only done for the area outside of the fovea, which is a blind spot for motion perception and therefore is suitable for highly detailed visual tasks like reading.

The advantages of autonomous driving are diminished with motion sickness and may be even safety-critical when the autonomous vehicle first induces a person with motion sickness and then requires this person to take full control. To combat

FIGURE 1.3: A-pillar display concept by Continental AG [Con18].

motion sickness, Diels and Bos [DB16] propose the following guidelines for cars: Maximum window surface areas, minimal visual obstruction by A-pillars and seats with a sufficient height to look out of the vehicle to increase the perception of visual motion cues. It is even proposed to integrate displays in the vehicle interior like the door cards of the floor to show congruent visual motion. Figure 1.3 shows an image of a concept how a transparent A-pillar combined with driver eye tracking could look like.

As described before, we can exchange the vehicle model freely with a 3D scan of the scene in the VR HMD case. With an invisible vehicle hull in VR, the vehicle motion is made more obvious due to this "virtual convertible" effect. Furthermore, an AR HMD would also allow one to see-through the vehicle hull, *e.g.* making the hood transparent or removing the A and B-pillars. These are all measures that can massively improve the comfort of passengers during an autonomous ride. Such a system could also improve driver awareness and would allow a full head-up display (HUD) system with 3D occlusions in scenes, which may replace traditional screens. This also results in a less claustrophobic interior and would enable other car design choices.

Note that besides the technical challenges, there exist other hurdles for an in-series use of HMDs. For example, it is unclear whether the current HMD design can be used along with a front airbag, which may lead to difficulties in future vehicles' certification processes.

A reconstructed 3D scene for an immersive experience via HMDs would have to be dense, efficiently computed and must provide a high texture quality as the scene can be seen from multiple view points. The reconstruction needs to happen live on an in-vehicle system and needs to be rendered with a frame rate of ca. 90 Hz to achieve an immersive experience on modern HMDs. This puts great constraints on

computational resources and may restrict the use of too complex 3D reconstruction techniques. Furthermore, an in-series solution would add additional requirements in respect to hardware costs, integration and computational power. Generally, a dense 3D scene reconstruction has additional applications in intelligent robotics such as navigation, obstacle avoidance and scene understanding.

For this task, we propose to use a single front-mounted stereo camera on a driving vehicle along urban scenery. A stereo camera is already used in modern vehicles for driver assistance, *e.g.* pedestrian detection and traffic sign detection. This means that it is already built-in and necessarily calibrated. A stereo camera itself is widely researched and available. Compared to Lidar (light detection and ranging) or other sensors, it is also a relatively cheap sensor hardware-wise.

The target application of this thesis is the live reconstruction of the 3D scene within the FoV of vehicle occupants, which might be used for immersive Augmented and Mixed Reality use cases. We assume that an accurate pose estimation of the vehicle is already available from the regular AR feature or other systems (navigation, driver assistance), plus an accurate calibration from the vehicle coordinate system to the left camera of the stereo camera pair. Generally, the camera movement is limited to the vehicle movement, which is advantageous for the pose estimation: Pitch and roll angles of the vehicles are very constrained compared to *e.g.* aerial vehicles because the vehicle will not flip or roll over in normal situations. Also, the vehicle generally moves only forwards. Due to this movement along the optical axis, static objects in the scene (as seen from the camera) will only get closer in depth over multiple frames until they leave the FoV of the camera. The stereo accuracy highly depends on the depth itself and closer points can be reconstructed more reliably than points further away. For instance, while driving forwards, the depth accuracy of an observed scene point will change more drastically in a front-facing scenario than in a side-facing scenario. In such a side-facing scenario, the same scene point is seen at roughly the same depth, at *e.g.* a building facade. We can also reobserve scene points in multiple frames more often in a front-facing scenario, which is also advantageous for the tracking robustness of visual odometry approaches.

Typical high-detail 3D reconstruction scenarios consist of indoor scenes like office spaces and using RGB-D cameras (see Newcombe *et al.* [New+11]). The measurements are taken from close distance, while in our urban scenarios scene points can have a true depth of $100\,\mathrm{m}$ or up to several kilometers on highways. Furthermore, RGB-D cameras tend to not work under outside conditions due to infrared light interference from the sun.

Usually, an urban scene consists of a U-shaped city canyon: The sides are facades of buildings, the bottom is the road and the top is open because of the sky. Urban scenes are cluttered with dynamic objects, *e.g.* vehicles, cyclists and pedestrians, which are challenging to reconstruct in 3D. Furthermore, the scene content may include vegetation such as trees and bushes, which provide immensely complex 3D geometry to reconstruct compared to *e.g.* an office chair. Also, the scenes feature

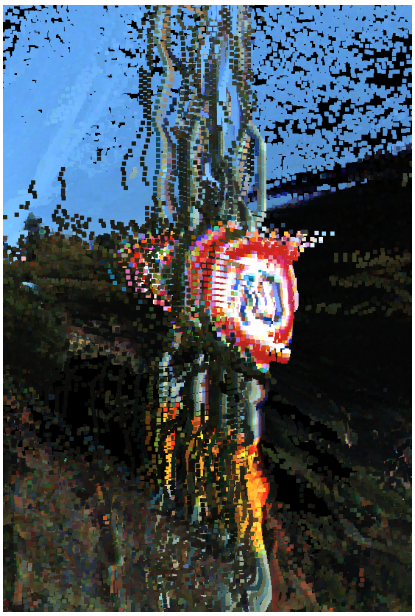


FIGURE 1.4: Visualizing every reconstructed point of the stereo matcher ELAS [GRU10] with ground truth poses on KITTI VO 04.

FIGURE 1.5: Reconstruction result of our approach with PSMNet [CC18] on KITTI VO 02.

highly salient objects like traffic signs, which have to be adequately reconstructed due to their recognizability and visual importance in immersive applications. Naively combining stereo measurements from multiple frames may result in a traffic sign appearing duplicated in the scene as shown in Figure 1.4, which needs to be prevented. Figure 1.5 shows an example reconstruction result of our approach over multiple frames. The scene is rendered as multiple meshes. It features no duplicated scene content and looks very consistent with the exception of sky pixels.

For stereo matching itself, the scene content is usually more challenging than an indoor office space. Reflecting surfaces like windows, vehicle paint and glass facades are commonplace and may provide challenges for conventional stereo matching methods. Also, large untextured regions such as house facades or the sky, as well as sunlight glare may provide additional difficulties.

1.2 Contributions

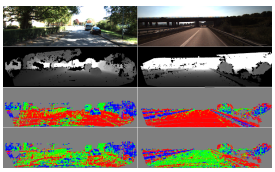
Our contributions can be summarized as follows:

- Analysis of the (temporal) fusion of stereo measurements from multiple frames, including the derivation, evaluation and comparison of a 1D and a 3D probabilistic filter on synthetic data and its limitations. Our filters outperform other naive fusion approaches on simulated stereo measurements.
- A novel 3D dense reconstruction framework for a front-mounted stereo camera on a vehicle, which models each measurement with a separate filter and is able to distinguish multiple depth layers, as well as to remove dynamic parts of the scene in the active region. The reconstruction framework features a point-based reconstruction approach with a 1D extended information filter, as well as a mesh-based reconstruction approach with a 3D extended Kalman filter.
- A very fast, novel visualization method suitable for immersive applications to quickly render the surrounding urban scene. It features depth image triangulation and our new concept of tube meshes using only images and camera poses as the input to the GPU.
- The generation of a virtual sky background seeded from detected sky pixels in the camera image using a cubemap and GPU shaders.
- An extended evaluation on real and synthetic data of multiple reconstruction approaches in regards to depth accuracy, depth structure metrics and view prediction error and their limitations, as well as ablation studies for our framework and a qualitative showcase on multiple datasets with different stereo baselines. In the evaluation, our approach is able to improve the depth accuracy compared to the input and outperforms other approaches in terms of the view prediction error metrics.

Figure 1.6 is a list of our publications that may therefore have overlapping content with this thesis.

In-Car 6-DoF Mixed Reality for Rear-Seat and Co-Driver Entertainment
Jonas Haeling, Christian Winkler, Stephan Leenders, Daniel Keßelheim, Axel Hildebrand, Marc Necker
IEEE VR 2018 (Demo). [Hae+18]

Towards Immersive Stereo Vision from a Mobile Platform
Jonas Haeling, Andreas Schilling, Marc Necker
ICMV 2018. [HNS19b]



Calibrating Depth Sensors with a Genetic Algorithm
Jonas Haeling, Andreas Schilling, Marc Necker
Technical Report. [HNS19a]

Dense Urban Scene Reconstruction using Stereo Depth Image Triangulation
Jonas Haeling, Andreas Schilling, Marc Necker
ICMV 2019. [HNS20]

FIGURE 1.6: List of our publications.

1.3 Notation

Throughout this thesis, we use a consistent notation for mathematical symbols. Matrices are displayed as boldface uppercase letters, while vectors appear as boldface lowercase letters. Scalars are regular letters and may be uppercase if they represent a 1×1 matrix (*e.g.* a 1D Jacobian) or the stereo baseline B (convention).

1.4 Structure of this Thesis

This thesis is structured as follows: In Chapter 2 the necessary background to follow along with this thesis is explained, while Chapter 3 summarizes related research. In Chapter 4 we analyze the effects of stereo fusion. Chapter 5 integrates the resulting insights in a 3D reconstruction framework and details our proposed visualization methods. Quantitative and qualitative results are subsequently presented in Chapter 6. Finally, Chapter 7 summarizes this thesis and gives an outlook for further research.

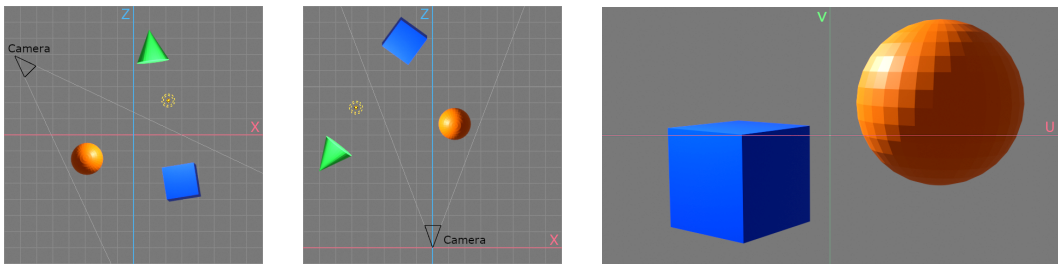
Chapter 2

Background

In this chapter, we will present the necessary background to follow along with this thesis.

2.1 Monocular Camera

A monocular camera is a passive sensor. Instead of sending a signal out, a camera will only receive incoming light. It can be modeled as a pinhole camera, in which the light enters a dark box through a very small opening. The light hits a light-sensitive sensor consisting of rectangular cells in the order of millions. These cells are illuminated for a fixed amount of time (shutter time) and a projected and discretized grayscale image of the outside can be measured by the image sensor. The extent of the projected image depends on the focal length, which is the distance of the optical center to the image plane, and the sensor size. These limits of the camera determine its FoV and therefore also the view volume. In computer graphics, this view volume is referred to as the view frustum, which is limited in Z-direction (along the optical axis) by a near and far plane to skip the rendering of 3D entities outside the view frustum (culling).



(a) World space. (b) Camera space. (c) Image space.

FIGURE 2.1: The different coordinate systems used for the perspective projection in a 3D scene with a single light source rendered with Blender [Ble18]. Axes are visualized and the world and camera space are seen from the top.

To transform 3D world points to 2D image space we need to apply a perspective projection. For this, we define three coordinate systems as illustrated in Figure 2.1. First, there is the world space. In it, we define the 3D camera space, which is a local coordinate system placed on the camera center, where the Z axis is aligned with the optical axis and the X and Y axes are parallel to the image plane. Besides this,

there is the 2D image space, which describes point coordinates on the image plane in pixels with the center at the principal point (c_x, c_y). The aforementioned perspective projection is the transformation from camera space to image space.

Points in camera space are projected onto the same depth plane, *i.e.* the image plane, thus losing a dimension from 3D to 2D. To handle these projections, homogeneous coordinates are conventionally used, which add another dimension to 3D coordinates. This helps us write the transformation in a single matrix. The projection of a 3D point $\mathbf{p_c}$ in camera space to a 2D point $\mathbf{p_i}$ in image space can be modeled as follows with the pinhole camera model, introducing the intrinsic matrix $\mathbf{K}$:

$$\mathbf{K}\begin{bmatrix}\mathbf{p_c}\\1\end{bmatrix} = \begin{bmatrix}f_x & 0 & c_x & 0\\0 & f_y & c_y & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_c\\y_c\\z_c\\1\end{bmatrix} = \begin{bmatrix}f_x x_c + c_x z_c\\f_y y_c + c_y z_c\\z_c\\1\end{bmatrix} = z_c\begin{bmatrix}u\\v\\1\\\frac{1}{z_c}\end{bmatrix} = z_c\begin{bmatrix}\mathbf{p_i}\\1\\\frac{1}{z_c}\end{bmatrix}, \tag{2.1}$$

where f_x and f_y denote the horizontal and vertical focal length in pixels, respectively, while c_x denotes the horizontal center of the image space and c_y the vertical center. Note that we introduced the image space coordinates u and v. Analogously, we can backproject a 2D pixel at (u, v) from image space to camera space if its depth z_c is known:

$$\mathbf{K^{-1}} z_c\begin{bmatrix}u\\v\\1\\\frac{1}{z_c}\end{bmatrix} = \begin{bmatrix}\frac{1}{fx} & 0 & -\frac{c_x}{fx} & 0\\0 & \frac{1}{f_y} & -\frac{c_y}{fy} & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}f_x x_c + c_x z_c\\f_y y_c + c_y z_c\\z_c\\1\end{bmatrix} = \begin{bmatrix}x_c\\y_c\\z_c\\1\end{bmatrix} \tag{2.2}$$

Additionally, cameras are usually calibrated to alleviate distortion artifacts like tangential distortion. For that, a calibration procedure using *e.g.* a checkerboard pattern on a board with exactly known dimensions is used.

The coordinate system transformation from 3D world coordinates to 3D camera coordinates is determined by the extrinsic parameters. The extrinsic parameters are represented by a translation vector $\mathbf{t}$, which transforms the world origin to the camera center, and a rotation matrix $\mathbf{R}$, which rotates the world coordinate system to the camera orientation. Here, we insert both $\mathbf{R}$ and $\mathbf{t}$ in separate 4×4 identity matrices so that $\mathbf{t}$ corresponds to the unrotated camera translation. Now, a 3D world point $\mathbf{p_w} = \begin{bmatrix} x_w & y_w & z_w \end{bmatrix}^T$ can be projected to a 2D point $\mathbf{p_i} = \begin{bmatrix} u & v \end{bmatrix}^T$ in image space as follows:

$$\mathbf{P}\begin{bmatrix}\mathbf{p_w}\\1\end{bmatrix} = \mathbf{K}\begin{bmatrix}\mathbf{R_{3\times3}} & \mathbf{0_{3\times1}}\\\mathbf{0_{1\times3}} & 1\end{bmatrix}\begin{bmatrix}\mathbf{I_{3\times3}} & \mathbf{t_{3\times1}}\\\mathbf{0_{1\times3}} & 1\end{bmatrix}\begin{bmatrix}x_w\\y_w\\z_w\\1\end{bmatrix} = z_c\begin{bmatrix}u\\v\\1\\\frac{1}{z_c}\end{bmatrix} \tag{2.3}$$

z_c again denotes the depth of the 3D point in camera space. We call $\mathbf{P}$ the projection matrix. Here, the camera pose, *i.e.* the position and orientation of the camera in world space, can be derived from the extrinsic parameters by substituting $-\mathbf{t}$ for $\mathbf{t}$ and $\mathbf{R}^\mathbf{T}$ for $\mathbf{R}$.

2.2 Stereo Camera

Stereo cameras essentially work similarly to the human eyes. A stereo camera setup consists of two cameras arranged in such a way that they both observe similar parts of the scene. Usually, their optical axes are parallel or close to parallel before the rectification step. One can estimate the stereo baseline of these cameras, which is the distance between the two optical centers, via stereo calibration procedures. The stereo calibration makes use of the epipolar geometry, which allows us to find correspondences of 3D rays from the center of one camera in the other camera. The stereo calibration includes the rectification of both images, which means that both images are transformed in such a way that these aforementioned rays now lie along a single pixel row in the other image. This simplifies the search for correspondences between the left and right stereo image from a 2D problem to a 1D problem. Correspondences are needed to triangulate scene points and thus to estimate depths.

2.2.1 Stereo Matching

The task of stereo matching is to find correspondences between the left and right rectified images. *Local* stereo matching approaches consist of finding correlated image patches between the pixel in the first image and the pixel in the same row in the second image. For that, a variety of different cost metrics are computed to determine the best match. Often optimization techniques such as linear programming are used. Some techniques incorporate more *global* (or *semi-global*) contexts, which allow for more globally consistent and smooth estimations. Most modern techniques try to estimate disparities at a subpixel resolution accuracy, *e.g.* by fitting a parabola into the cost volume to locate the cost minimum more accurately.

2.2.2 Depth from Stereo and Disparity

Having found a correspondence, we can compute the horizontal pixel difference from the image centers, which is called disparity. Disparity is the depth representation in image space and is a parallax effect. Pixels that have a higher disparity are closer to the observer, and lower disparity pixels are farther away. For example, a far away mountain top may have a really low true disparity value, *i.e.* the mountain top is almost depicted at the same pixel location in both images. This is similar to human stereo vision.

In Figure 2.2, we show a typical stereo situation with a left and right camera with respective camera centers $\mathbf{O_l}$ and $\mathbf{O_r}$, which both observe the scene point $\mathbf{P}$. Note

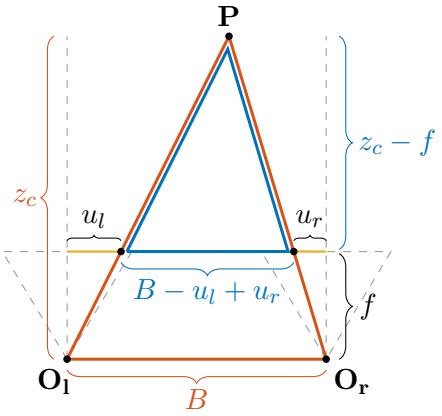


FIGURE 2.2: Top-down view of a stereo camera rig triangulating a point **P** in camera space for depth estimate z_c.

that here all variables share the same unit for illustration purposes. Our goal is to estimate the depth z_c of point **P**. For that, we use the similarity of the big orange triangle $\overline{\mathbf{O_l O_r P}}$ and the small blue triangle, which consists of the two projections of the point **P** and **P** itself. Thus, we can write:

$$\frac{B}{z_c} = \frac{B - u_l + u_r}{z_c - f} \tag{2.4}$$

$$\Rightarrow z_c = \frac{Bf}{u_l - u_r} = \frac{Bf}{d} \tag{2.5}$$

Note that the signs of u_r and u_l have flipped in the last equation. The disparity can now be defined as $d = u_l - u_r$, which allows us to estimate depth from stereo images. Only the focal length f and stereo baseline B geometrically determine the depth of a stereo match. We call an image with a per-pixel estimated depth a depth image, which can be calculated from a disparity image via Equation 2.5. They are also known as depth maps and disparity maps, respectively.

Stereo matching faces certain challenges and limitations. Homogeneous textures are notoriously difficult for stereo matching algorithms, because there is not enough texture variation to determine where exactly the correct match should be. For example, a stereo matcher that searches for correlations of 3×3 image patches will find a lot of matching candidates in a larger homogeneous textured area since they are (almost) identical. In a similar vein, repeated textures may induce artifacts since each repetition may pose as a great matching candidate locally. Furthermore, as stereo cameras are passive sensors, their performance is limited under low-light conditions. Also, they see the world from two different perspectives, which has to result in non-identical pixel intensities at *e.g.* specular surfaces. Additionally, mirrors and glass surfaces may provide depth cues for multiple depth layers due to their reflectance, *i.e.* the position of the actual glass and the reflected scene content, which provides ambiguous stereo matching results. Thus, a good stereo matching algorithm has to handle these difficult cases in which the pixels are not a trivial match.

2.3 3D Reconstruction

2.3.1 Warping

To temporally combine measurements from a sequence of disparity images, we need to transform pixel locations from one frame to another frame. This is called warping in the 3D reconstruction context. It is possible because both the depth of a pixel in one frame and the camera poses of the two frames are known. This allows us to backproject 2D pixels from one camera to 3D world space and then project the 3D world point to another camera image. Thus, to warp a pixel $\mathbf{p_A}$ from camera A with depth z_{c_A} to camera B (pixel $\mathbf{p'_A}$), we have to multiply it by the inverse projection matrix of A ($\mathbf{P_A^{-1}}$) and then use the projection matrix of B ($\mathbf{P_B}$):

$$\mathbf{p'_A} = \mathbf{P_B}\mathbf{P_A^{-1}}\mathbf{p_A} = \mathbf{P_B}\mathbf{P_A^{-1}} z_{c_a} \begin{bmatrix} u \\ v \\ 1 \\ \frac{1}{z_{c_a}} \end{bmatrix} \tag{2.6}$$

2.3.2 Point Cloud

The corresponding (color) pixels of a disparity or depth image can be backprojected to 3D as mentioned before. Given a sequence of images or just a single frame, this forms a point cloud. In its simplest form, a point cloud consists of individual points in a list, which can be rendered in 3D. Points have no volume and area mathematically, but point primitives can be rendered in 3D engines like OpenGL as small squares ("quads") that are aligned to the screen with adjustable size. A similar point-based rendering method is called splatting, which models points *e.g.* as oriented ellipses (quads with transparency).

2.3.3 Rendering Pipeline and Meshes

Modern GPUs have a programmable rendering pipeline. Shaders are programs that run on the GPU for certain stages of the rendering pipeline. A general use case is the following: The pipeline takes as input a list of 3D points with attributes, an index list, which corresponds to triangles, and also textures (2D images) as well as camera transformation matrix. The output is a rendered 2D image of the textured geometry as seen from the camera. Programmable stages of the pipeline can include the vertex shader, geometry shader and fragment shader, which we employed in this work and describe in the following.

The vertex shader can operate on the individual vertices of the geometry. A common task for the vertex shader is to project the world 3D geometry to the current virtual camera. Then, the geometry shader allows operations on whole triangles, which allows for *e.g.* the tessellation of 3D geometry (subdividing triangles for more details). The fragment shader is the next stage in the pipeline. Here, the input is

already rasterized, which means that the mathematical description of the triangles is transformed to discrete 2D pixels (fragments). Here, a common task is to sample textures at the interpolated UV coordinate (pixel coordinates) for the color of this fragment combined with lighting effects according to its normal, which improves the visual fidelity compared to the potentially very coarse interpolation possible at vertex level. Figure 2.1c shows an example of a render of a sphere with this coarse flat shading.

The list of triangles and indices is called a mesh. Compared to a point cloud, it enables a potentially watertight reconstruction. It consists of 3D triangles, which are rasterized to pixels. Using highly detailed textures with UV mapping allows us to use coarser meshes as the texture still appears to be highly detailed.

2.4 Kalman Filter

Kalman filters are a class of probabilistic filters, which use an iterative loop of propagation and measurement to improve the estimates of a series of measurements. The state is modeled with a (multi-variate) normal distribution, which means that Kalman filters are unimodal and can only represent a single state estimate.

A snapshot of a Kalman filter consists of a state, which is propagated to a new state according to a propagation model, as well as a covariance matrix which captures the estimated uncertainty of the state. In the propagation step, the covariance matrix is also propagated. This means generally that some sort of uncertain movement occurred (modeled via a control variable), which increases uncertainty in the covariance matrix.

In the update or measurement step, the Kalman filter essentially computes how much it should trust the new measurement and its covariance compared to the propagated values via the Kalman matrix $\mathbf{K}$. Then it fuses the propagated state and propagated covariance matrix with the new observation to a new state and covariance matrix accordingly. Here, the uncertainty is generally reduced as a new observation is integrated.

As an illustrative example, imagine a robot, which wants to know its 1D distance to a wall. The robot measures the distance via some sensor and models the uncertainty of this measurement as a normal distribution with certain parameters (measurement noise). Then the robot moves, which propagates this distribution to a new distance as its new state. At the same time, the robot knows approximately how it moved according to some model (process noise), which introduces more uncertainty in the propagated distribution. Then, it can update its state by measuring again, leading to a reduced variance in the distribution and the new state estimate.

The Kalman filter only works with linear propagation and observation models. For non-linear models, we can use an extended Kalman filter (EKF). An EKF linearizes the non-linear models at the current state locally with Jacobians (1^{st} order Taylor

expansion). EKFs are a widely researched topic and various other extensions exist that try to improve the handling of non-linearity.

Here, we show the general formulas of an extended Kalman filter and an extended information filter (EIF) according to Thrun *et al*. [TBF05]. An information filter is the dual problem formulation to the Kalman filter. It uses the inverse of the covariance matrix instead, which is called the Fisher information matrix or simply information matrix. Problems are formulated in the dual form because the complexity of the propagation and update steps are different, which might prove beneficial for an efficient implementation.

We renamed the control variable $\mathbf{u_t}$ in the original EKF formulation to $\mathbf{c_t}$ to prevent name clashes with image space state variables. We also substituted the information vector representation in the EIF with the EKF state to highlight similarities. The equations can be written as follows (according to Thrun *et al*. [TBF05] with the mentioned changes):

	Extended Kalman Filter	Extended Information Filter
Propagation	$\bar{\boldsymbol{\mu}}_\mathbf{t} = g(\mathbf{c_t}, \boldsymbol{\mu}_\mathbf{t-1})$	$\bar{\boldsymbol{\mu}}_\mathbf{t} = g(\mathbf{c_t}, \boldsymbol{\mu}_\mathbf{t-1})$
	$\bar{\boldsymbol{\Sigma}}_\mathbf{t} = \mathbf{G_t}\boldsymbol{\Sigma}_\mathbf{t-1}\mathbf{G_t^T} + \mathbf{R_t}$	$\bar{\boldsymbol{\Omega}}_\mathbf{t} = \left(\mathbf{G_t}\boldsymbol{\Omega}_\mathbf{t-1}^{-1}\mathbf{G_t^T} + \mathbf{R_t}\right)^{-1}$
Update	$\mathbf{K_t} = \bar{\boldsymbol{\Sigma}}_\mathbf{t}\mathbf{H_t^T}(\mathbf{H_t}\bar{\boldsymbol{\Sigma}}_\mathbf{t}\mathbf{H_t^T} + \mathbf{Q_t})^{-1}$	$\boldsymbol{\Omega}_\mathbf{t} = \bar{\boldsymbol{\Omega}}_\mathbf{t} + \mathbf{H_t^T}\mathbf{Q_t^{-1}}\mathbf{H^T}$
	$\boldsymbol{\mu}_\mathbf{t} = \bar{\boldsymbol{\mu}}_\mathbf{t} + \mathbf{K_t}(\mathbf{z_t} - h(\bar{\boldsymbol{\mu}}_\mathbf{t}))$	$\boldsymbol{\mu}_\mathbf{t} = \boldsymbol{\Omega}_\mathbf{t}^{-1}(\bar{\boldsymbol{\Omega}}_\mathbf{t}\bar{\boldsymbol{\mu}}_\mathbf{t} + \mathbf{H_t^T}\mathbf{Q_t^{-1}}$
	$\boldsymbol{\Sigma}_\mathbf{t} = (\mathbf{I} - \mathbf{K_t}\mathbf{H_t})\bar{\boldsymbol{\Sigma}}_\mathbf{t}$	$\cdot [\mathbf{z_t} - h(\bar{\boldsymbol{\mu}}_\mathbf{t}) + \mathbf{H_t}\bar{\boldsymbol{\mu}}_\mathbf{t}])$

Starting from the propagation equations, the state vector at time t is represented by $\boldsymbol{\mu}_\mathbf{t}$. g is the potentially non-linear state transition function, which, besides the previous state $\boldsymbol{\mu}_\mathbf{t-1}$, requires a control vector $\mathbf{c_t}$ to predict the propagated state $\bar{\boldsymbol{\mu}}_\mathbf{t}$. $\boldsymbol{\Sigma}_\mathbf{t}$ is the covariance matrix at time t and $\boldsymbol{\Omega}_\mathbf{t}$ the information matrix at time t. The Jacobian matrix $\mathbf{G_t}$ consists of the partial derivatives of g with respect to the previous state $\boldsymbol{\mu}_\mathbf{t-1}$. $\mathbf{R_t}$ is the process noise matrix, which models additional uncertainty induced by the state transition.

Similarly, in the update equations, $\mathbf{Q_t}$ is the measurement noise matrix and represents the uncertainty of the new measurement $\mathbf{z_t}$. h is the measurement function and $\mathbf{H_t}$ is the observation matrix, which transform the measurement space to the true state space.

2.5 Augmented and Virtual Reality

Augmented and virtual reality can be conceptually placed on a reality-virtuality continuum (cf. Milgram *et al*. [Mil+95]) as depicted in Figure 2.3. The left side symbolizes the real environment, while the right side shows a purely virtual environment. Enriching the real world with virtual elements is augmented reality, while enriching

the virtual world with real elements is called augmented virtuality. Milgram *et al*. [Mil+95] call the in-between spectrum Mixed Reality, while we would only consider systems with approximately equal real and virtual parts Mixed Reality like the Microsoft HoloLens 2 [Mic19]. Even though there is no clear-cut definition for the more recent term Extended Reality, it is now often used to describe the whole spectrum instead of MR.

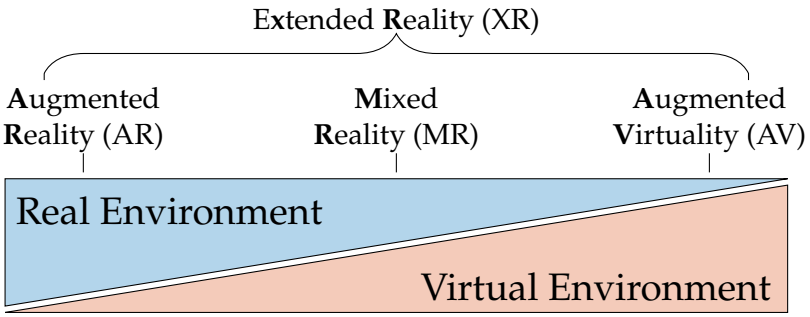


FIGURE 2.3: Reality-virtuality continuum from Milgram *et al*. [Mil+95] with own extension.

In Section 1.1 we already discussed AR and VR HMDs in the automotive context. Using an HMD in a moving platform, which is one part of our described target application, has an additional challenge compared to room VR setups. Usually, the HMD tracking setup consists of an optical tracker and an IMU (inertial measurement unit) inside the headset. The optical tracker only detects head movement when it is mounted inside the car. The IMU measures the head movement relative to the world including vehicle forces. The forces, which are induced by the vehicle, *e.g*. in curves while braking and accelerating, are added on top of the forces created by head movement alone. This creates contradictory evidence during the sensor fusion in conventional tracking approaches inside a vehicle, similar to why motion sickness is hypothesized to occur in humans. In this case, the IMU observes the relative head movement to the world while the optical sensor in the vehicle only picks up the head movement relative to the vehicle. This results in a loss of HMD tracking.

Chapter 3

Related Work

3.1 Overview of Related Research Areas

In this thesis, we focus on 3D reconstruction with a single stereo camera and in this chapter, we give an overview of related research areas and classes of algorithms. They all have in common that they are camera-based and incorporate an estimation of the camera pose. Major research areas are:

- **Simultaneous Localization and Mapping** (SLAM): SLAM has its origin in robotics and is used to estimate the robots movement in a yet unknown environment. SLAM approaches fuse sensor measurements to estimate their own position and at the same time estimate a map of the environment iteratively in real-time. Loop closures help in creating a better map and pose estimate by detecting previous parts of the scene and optimizing it to a consistent new pose and map estimate. Thrun *et al*. [TBF05] provide an overview of basic SLAM concepts and algorithms.
- **Visual Odometry** (VO): VO is the task of estimating the trajectory of an imaging device like a monocular camera (mono camera) or a stereo camera. A sequence of images is fed iteratively to the algorithm, which estimates a new camera pose for each image based on its history. VO can be part of a SLAM system. Stereo cameras compared to mono cameras have the advantage of having no scale drift in VO because of the fixed stereo baseline. Additionally, VO itself has the advantage that the pose is perfectly time synchronized to the image (disregarding shutter effects). Planz [Pla19] examines the use of open-source stereo VO for our purpose.
- **3D Reconstruction**: 3D reconstruction encompasses the recovery of the projected 3D scene as a 3D model. It can be sparse, which means that only some confident feature points are extracted, or dense, which has a reconstruction resolution in the order of magnitude of the input pixels. Here, we talk about dense reconstruction when we refer to 3D reconstruction.
- **Multi-view Stereo** (MVS): MVS is similar to 3D reconstruction and can be understood as stereo matching using more than two cameras in space or time. A typical use case would be a $360°$ motion capture or 3D scanning setup.

- **Structure from Motion** (SfM): SfM is another photogrammetry technique, which has overlap with 3D reconstruction and possibly MVS or SLAM. SfM usually encompasses a collection of high-resolution images from differently positioned cameras to reconstruct a whole 3D scene sparsely offline. In contrast to SLAM, SfM generally does not have to be real-time capable. SfM optimizes the whole collection of images to estimate the camera poses and even intrinsic parameters when using different cameras to extract 3D structures.
- **Virtual View Synthesis** (VVS): VVS is an interpolation technique between images to create a new virtual view. It implicitly reconstructs a 3D scene and uses different interpolation techniques to hide occluded scene content. It is also known as free viewpoint synthesis or video. VVS is used in sporting events to create smooth camera perspective transitions or for potentially enabling eye contact in video conferencing (see Dumont [Dum15]).

3.2 RGB-D Cameras, Voxels and Splats

Active sensors like the Microsoft Kinect have been used for online 3D reconstruction of limited indoor scenes. Newcombe *et al*. [New+11] present seminal work in this area, featuring truncated signed-distance functions (TSDF, cf. Curless and Levoy [CL96]) to reconstruct a small 3D scene with high quality. A TSDF consists of a 3D voxel grid, whereas each voxel has a signed distance to the nearest surface, *e.g*. positive or negative to the zero-crossing as seen from the camera. To render a TSDF, the zero-crossings of the implicit surface have to be computed for each pixel ray via an iterative approximation algorithm (volume ray casting). The static 3D voxel grid might need to be reallocated dynamically to cover other parts of the scene.

The approach by Newcombe *et al*. [New+11] features an incremental fusion of measurements into a 3D voxel-based grid and a camera pose tracking at the same time. Unfortunately, RGB-D sensors generally do not work well in outdoor scenarios. Since voxel-based approaches place a 3D grid into the scene, there needs to be an efficient mechanism to extend the reconstruction beyond the already reconstructed volume. This limited early approaches to only small scenes such as office spaces due to great memory requirements. Still, voxel-based approaches may lack visual fidelity since their level of detail can be coarse compared to a highly detailed scene. Increasing the resolution of the voxel grid would greatly increase the memory requirements, again. Also, color averaging in voxels may provide poor texture quality.

Another approach with a RGB-D camera is presented in Keller *et al*. [Kel+13], which tries to diminish the weaknesses of voxel approaches by using a point-based fusion approach. Points of an active part of the scene are reprojected to the new camera frame and fused according to some criteria. For rendering, they use point-based rendering via splats instead of an implicit surface. Splats are essentially 3D oriented textured quads, which may require normal information and cover a certain area of that particular point. This enables rendering of an appropriately sized and

oriented primitive instead of a 3D point with no volume. With splatting, even points may approximately cover surfaces.

Some challenges exist for splats: First, they cannot completely compensate for sparse input clouds. Second, they require special handling at edges of scene geometry so that splats do not overextend. Third, compared to textured 3D meshes, the UV handling is more complex to achieve comparable results.

3.3 Image Triangulation

Pixels on an image sensor are mostly squares or at least rectangular. Images form a regular grid, where each square cell represents a pixel. Each cell can be trivially represented by two triangles which fill out this area. We call this and other variants image triangulation. An example triangulation of pixel cells is presented in Figure 3.1. A 5×5 pixel patch of the top of the bird house roof is magnified to show a potential triangulation of the grid.

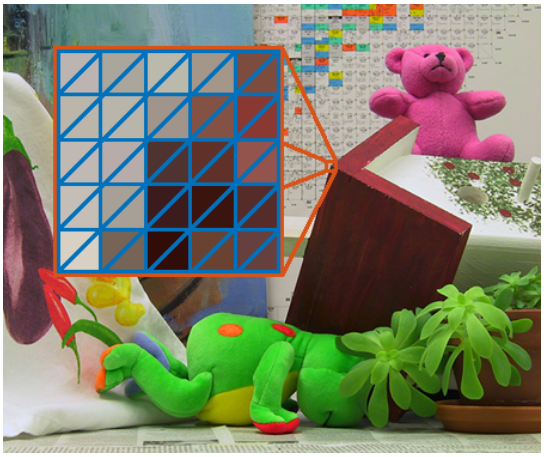


FIGURE 3.1: Example of image triangulation concept for a 5×5 image patch on a Middlebury dataset [SS02] image.

A similar method is known as heightmapping, which is a common 3D graphics technique. A heightmap can be thought of as a regular triangle grid in 2D lying on the ground floor, in which each vertex gets displaced from the floor to 3D by a certain height sampled from a 2D texture. For example, one could use an input 8 bit gray-scale image of 512×512 pixels to generate a mesh with $262\,144$ surface points at 256 height levels. The concept of heightmapping is widely used in rendering surface elevation data in 3D or terrain rendering in video games, usually using a tessellation shader for smoother surfaces.

Image triangulation is also used in other varying contexts in 3D graphics or computer vision. For example, in the context of virtual view synthesis, Tola *et al*. [Tol+09] place the vertices on the pixel centers and use a depth map to displace the mesh to 3D. A similar approach has also been used in the 3D reconstruction of foreground objects: Alexiadis *et al*. [AZD13] use four different RGB-D cameras

to individually displace a 2D depth image with depth discontinuity constraints to 3D. In a wholly different context, 2D screen space meshes are used to render 3D particles in Müller *et al*. [MSD07]. Particle silhouettes are extracted in 2D and are then backprojected to 3D, which is an efficient way to render fluids, for example.

For stereo matching, there are approaches that triangulate pixels in the 2D depth image, but not on a per-pixel basis. Geiger *et al*. [GRU10] and Pillai *et al*. [PRL16] both detect feature point correspondences in image space to construct a Delaunay triangulation in image space. The stereo matching is densified by sampling disparities from the spanning mesh. Staib [Sta18] proposes to split and merge vertices of a coarse triangulation of a RGB and depth image pair over multiple frames. Adding another vertex depends on the scene geometry measured in depth, normal and curvature information. Furthermore, Zhang *et al*. [Zha+15] partition the image into 2D triangular regions using polygonized superpixel segmentation. The resulting triangles have shared vertices, which can be displaced to 3D. Depth edges are split according to an extra optimization step to prevent too large triangles spanning depth discontinuities.

3.4 Stereo Matching and Filtering

Thanks to the work of Scharstein and Szeliski [SS02], the development of stereo matching algorithms is traditionally focussed on the performance in public benchmarks. They provide the Middlebury Stereo Evaluation webpage, which features a training and test set of various stereo camera images in different scenes. An example image can be seen in Figure 3.1. At the point of writing, the benchmark is in the 3rd edition and provides subpixel-accurate ground truth depth as detailed by Scharstein *et al*. [Sch+14]. Stereo matching metrics in benchmarks usually include metrics such as the ratio of bad pixels above a certain threshold, error quantiles, average errors, the density of the depth image and runtime performance.

In the automotive context and in the context of autonomous driving, the KITTI Vision Benchmark Suite (short: KITTI) [GLU12] provides a similar benchmark for stereo data as seen from a front-mounted stereo camera. It also features various measurements from other sensors such as a Lidar, which provides semi-dense 360° distance estimates. For stereo matching, there exists the 2012 version and also the 2015 version by Menze *et al*. [MG15], which features a multi-view sequence extension. The ground truth is generated by aggregating Lidar measurements. Additonally for KITTI 2015, virtual 3D vehicle objects are manually placed in the scene to get 3D data of the quite challenging vehicle surfaces. The KITTI benchmark also features a visual odometry benchmark, which has a generated ground truth using GPS and the IMU. The ground truth poses are published for the training data sequences.

We already introduced a stereo algorithm that we use in this thesis: ELAS (**E**fficient **LA**rge-scale **S**tereo Matching) from Geiger *et al*. [GRU10]. This approach represents a conventional stereo matching method, which works without a trained model. Recently, new deep learning approaches dominate the public stereo benchmarks.

To accomodate for this fact, we evaluate two state-of-the-art deep learning stereo matching algorithms, which are publicly available, as input to our framework.

First, we use PSMNet [CC18], which ranked first on both KITTI stereo benchmarks (2012 and 2015) at the time of its publication. PSMNet consists of a spatial pyramid pooling module, which aggregates local and global context information into a cost volume, and a 3D convolutional neural network with multiple stacked hourglass networks to regularize the cost volume.

The second and more recent approach is called GA-Net (**G**uided **A**ggregation **Net**) from Zhang *et al*. [Zha+19], which improves on the results of PSMNet on both KITTI 2012 and 2015 stereo benchmarks. They introduce new neural layers: First, a semi-global aggregation layer that approximates semi-global matching (see Hirschmüller [Hir08]). Second, it features a local guided aggregation layer, which is used to refine thin structures and object edges in the cost volume right before the disparity regression.

One advantage of deep learning approaches is that they can provide a dense depth estimation for all pixels. Even pixels that are not seen in either camera image of a stereo camera can be deducted from the context. One disadvantage of these trained algorithms is that they may have overfitted to the training scenery and benchmark metric. For example, it is not quite clear how those trained models would perform under different lighting or weather conditions or even different camera perspectives (generalizability). Also, traditional stereo camera algorithms have no learned biases, which helps them in more general stereo scenarios. Furthermore, the common benchmark metric of "bad pixels" may lead algorithms to optimize not necessarily towards perfect subpixel and bias-free accuracy, but in such a way that most of the pixels are good pixels, *i.e*. below the error threshold.

Probabilistic filters like Kalman filters are widely used in Robotics to integrate noisy measurements of multiple sensors into an improved estimate of a modeled state. In stereo, they appear in the context of 3D object motion estimation, scene flow and temporal fusion of stereo estimates (cf. [MKS89; LK90; Fra+05; SSM06; WC11; MK13]). In stereo SLAM, Schöps *et al*. [Sch+15] use an EKF for each pixel estimate with outlier filtering to update the estimated depth maps. The recent stereo SLAM framework LSD-SLAM [ESC15] proposes a direct SLAM algorithm, which works on the image intensities of the full image rather than feature points of scene geometry. Here, high-gradient pixels are used in combination with a Kalman filter to fuse stereo measurements temporally.

3.5 Urban Reconstruction

Here, we focus particularly on urban reconstruction from a moving vehicle, which may have different camera setups. A comprehensive survey, which also includes air-borne approaches for urban reconstruction, is presented in Musialski *et al*. [Mus+13].

An early urban reconstruction project is Cornelis *et al*. [Cor+08], which proposes to make simplified geometric assumptions for a high reconstruction speed with a single front-mounted stereo camera on a vehicle. These assumptions consist of approximating building facades in an U-shaped urban city scene by ruled surfaces parallel to the gravity vector. This simplifies the stereo matching for the facades to a single value per column in the image. The disadvantage is that scene objects have to be detected and virtually reinserted, as demonstrated via vehicles.

The *UrbanScape* project [Pol+15] proposes an automated urban 3D modelling system from the stream of four different cameras, all differently-angled and sidewards facing. Instead of stereo algorithms, sets of images were used for a plane sweeping algorithm on the GPU by Gallup [Gal11] as part of the project. A plane sweep in this context essentially tests different depth planes for each pixel of a certain view and chooses the depth at which the corresponding scene point is in focus of the input images with known camera poses. The set of images used to reconstruct parts of the scene can be freely chosen, *i.e*. the four camera images create a pool of images over time. This enables the variation of baseline and also focal length (via downsampling) to accomplish a fixed estimated accuracy for the reconstruction. Additionally, Gallup extended the plane sweeping by using slanted planes, which were detected beforehand. His assumption is that there are three main planes in an urban scene: The ground and the two wall planes of facades. Gallup also proposes a multi-layer heightmap representation for house facades, which are mostly vertical structures as seen from a vehicle on the road.

Using the KITTI vehicle, StereoScan [GZS11] combines the stereo matching algorithm ELAS with a visual odometry approach to 3D reconstruct a scene with grayscale cameras. They combine the estimated pose and computed depth image to backproject pixels to a 3D position. To lower memory usage and increase accuracy, they fuse measurements temporally by warping previous points to the next frame and only fuse and keep points that land on another consistent depth according to a depth threshold.

Very similarly, Alcantarilla *et al*. [ABD13] used KITTI data but relied on the ground truth poses of the training set of the KITTI visual odometry benchmark. They also use ELAS, but propose an even more aggressive filtering method for the reconstruction. Alcantarilla crosschecks past and future frames of the current frame geometrically and photometrically, by warping pixels from other frames to the current target frame. If all checks are successful, the points are combined to a single point. The fusion is weighted according to a pixel error measure.

Cigla *et al*. [CBM17] use a mixture of Gaussians to model pixels of ELAS. They also employ temporal fusion via warping to the current frame, which enabled them to improve the ELAS baseline results on the KITTI dataset. Furthermore, Pire *et al*. [Pir+18] use geometric checks during point fusion in a full stereo SLAM system.

All previously mentioned approaches (except Gallup [Gal11]) use a point cloud representation to represent the 3D reconstruction visually. Point cloud representations

have the disadvantage of thinning out when viewed up close due to sampling, which will appear as large holes in the rendering. In addition, naive geometrical checks between frames will limit the reconstruction generally only to close points. This is because these points are much more likely to be accurate enough due to the stereo error to fulfill *e.g.* a maximum depth difference threshold. As a consequence, the relative depth accuracy may improve due to a biased selection of near depths, while the overall density of the reconstruction is worsened.

There have been several undertakings in which a voxel-based approach to the mapping of urban scenery is incorporated. Sengupta *et al.* [Sen+13] model their urban scene geometry according to a TSDF. They further label this 3D reconstruction with semantically, but only consider an active $3 \times 3 \times 1$ grid of voxel volumes. When the vehicle passes the current grid, a new grid is allocated and initialized. Tanner *et al.* [Tan+16] try to overcome problems of voxel allocation using a hashing voxel grid data structure. They are able to provide a dense city-scale mapping system using a total general variation regularizer to remove spurious objects in the KITTI visual odometry dataset. Similarly, the InfiniTAM framework [Käh+15] is used by Bârsan in a dense stereo SLAM system with voxel hashing and TSDFs on the KITTI dataset. Additionally, dynamic objects are removed from the final reconstruction via semantic image segmentation computed by a convolutional neural network. Still, the voxel-based results of the aforementioned approaches are lacking texture detail in comparison to point-based approaches.

Furthermore, Wang *et al.* [WGS19] present a sparse visual SLAM system on the KITTI dataset, which uses superpixels to represent surfels (**surf**ace **el**ement**s**, splats). Similarly, Usenko *et al.* [Use+15] adapt the stereo LSD SLAM algorithm (see Engel *et al.* [ESC15]) to perform on KITTI in real-time on a single CPU, enabling a sparse reconstruction of a large-scale environment.

A fully automatic 3D reconstruction on the scale of a city is the *Varcity Project* by Vanhoey *et al.* [Van+17]. Their process combines aircraft images and street-level images from a special camera vehicle into a pipeline consisting of structure from motion, multi-view stereo and 3D meshing. Again, dynamic objects are excluded from the reconstruction and additional semantic information is provided.

Finally, as a whole alternative to stereo, Romanoni *et al.* [RFM17] use a Lidar combined with a camera to reconstruct a mesh. The camera images enable a photometrical refinement in an optimization step and also provide the texture of the reconstruction. Furthermore, Schönbein and Geiger [SG14] employ two omnicameras mounted on a vehicle to reconstruct urban scenes via stereo matching on rectified catadioptric stereo pairs.

3.6 In-Car AR/VR

The aforementioned augmented video series system of Mercedes-Benz [Grü13; Dai19] was the first system of its kind in an automotive context. As an extension of the system,

Schmid [Sch18] explores the concept of estimating occlusions with a monocular camera for automotive AR. Schmid proposes a semi-dense VO approach built on top of LSD SLAM (see Engel *et al*. [ESC15]) with a densifying step for a (pseudo) depth map and concludes with an analysis of mono and stereo comparison: The main advantage of stereo lies along the optical axis and in the fact that the stereo captures twice as many images as a mono camera. The mono camera has a singularity along the optical axis during forward camera movement, because the 3D triangulation requires observations from different 3D rays. Here, the 3D rays would approximately overlap, which makes the 3D estimation challenging or impossible in this scenario. A mono camera may still be used for (coarse) occlusion applications for scene points not close to the optical axis.

Next to the work on in-car AR, there have been various other projects that evaluate VR in a vehicle for passengers. Via a car diagnostic tool and an additional installed IMU, Hock *et al*. [Hoc+17] measure velocity and physical forces on the car. They use a rotational-only tracked HMD and mirror the movement of the vehicle to a virtual helicopter, which leads to an immersive VR experience. Similarly, McGill and Brewster [MB17] use a Samsung Gear VR with an extra smartphone as an IMU for the vehicle. They examine in-car VR motion sickness with multiple scenarios including 360° video. Their conclusion is that the right scenario depends highly on the individual and that positional tracking should be included since their solution experienced high angular drift.

As a demonstration for marketing purposes, Castrol Edge [Wal15] let race drivers race each other with VR HMDs in real cars through a virtual environment on a large empty area. Kodama *et al*. [Kod+17] use an electric car to navigate on a 1D track forwards and backwards in VR. To provide visual consistency, they subtract the measured real world forces of the tracked HMD.

AR has an often overlooked counterpart, which is called diminished reality (DR). Instead of augmenting or extending the current scene content, we may also remove parts of the scene. For instance, hiding unwanted advertisements or details from the user for extra focus. A recent and comprehensive survey for DR is presented in Mori *et al*. [MIS17]. AR see-through applications can also be considered as part of DR, as they are explicitly removing parts of the real scene to display the occluded background instead.

There have already been system concepts with a see-through effect: BMW Mini introduced their concept of AR glasses used inside a moving vehicle [Hob15]. The proposed use-cases included a see-through window through the hull of the vehicle to show occluded traffic participants or the sidewalk curb during a parking maneuver. Jaguar Rover [Lan14] demonstrated a transparent hood for offroad purposes from a single perspective using cameras in the front grill and in the side mirrors. Yoshida *et al*. [Yos+08] covered the interior of the vehicle with retroreflective cloth and used projectors to show a live video feed of the outside world on that surface.

Furthermore, there are systems that enable the vehicle driver to see through other

FIGURE 3.2: Our 6-DoF mixed reality in-car prototype (see Haeling *et al*. [Hae+18]). Top: HMD view, bottom left: Passenger with VR HMD, bottom right: Live video feed of front camera.

vehicles via a set of connected cameras, *i.e*. sharing live video feeds via vehicle-to-vehicle communication. One such system is Valeo's Xtra Vue [Val18] and another system is presented by Rameau *et al*. [Ram+16], in which the front car estimates a depth map via a front stereo camera. This depth map is warped to the rear vehicle's camera image by estimating the relative pose between the cameras.

We also developed an in-car VR prototype as an accompanying research project, which is however not the focal point of this thesis. Our system is shown in Figure 3.2. It features positional and rotational HMD tracking with six degrees of freedom (6-DoF) and mirrors the vehicle, its functions and also the street network. For example, users can see turn signals, the video of the AR camera in the central display and the current speed in the combination display. The street network is modeled with height data, but the rest of the scene can be chosen freely. Note that the street network serves as an anchor to the real world, which for example makes GPS offset errors very apparent compared to a virtual space or flight scene. We use the existing pose estimate of the AR system to get a highly accurate vehicle pose. Furthermore, we utilize an Oculus Rift CV1 [Fac19b] as our VR HMD without the positional tracking provided by the infrared tracker, but rely on the rotational-only tracking instead. This is because the standard tracking fuses both optical sensor and IMU, which we cannot separate in the consumer version. The conventional tracking fails due to additional vehicle forces measured by the IMU as mentioned before, which is why we used another infrared tracker for a positional estimate separately. Our approach delivers a high quality VR (or even mixed reality) in-vehicle experience with low motion sickness as reported by demonstration participants. Additional information can be found in Haeling *et al*. [Hae+18]. Recently, the start-up Holoride [hol19] presented a similar but commercial application of in-car VR. In the current demonstration,

participants are driven around fixed circuits in the real world while presented with an interactive fantasy setting in virtual reality.

Chapter 4

Analysis of Temporal Stereo Fusion

4.1 Introduction

The result of a stereo matching algorithm is a disparity map. Each disparity d can be transformed to a depth z non-linearly from pixels to meters via $z = \frac{Bf}{d}$ (see Section 2.2), where B is the stereo baseline in meters and f the (horizontal) focal length in pixels. The curve representing this function is a hyperbola. Thus, errors at lower disparities (*i.e.* far depths) will cause greater jumps in depth than the same disparity errors at higher disparities (*i.e.* close depths). This is illustrated in Figure 4.1, in which we vary the disparity by $1\,\mathrm{px}$ around a target pixel to see the stark differences in depth (as visualized by the height of the filled rectangles). Similarly, in Figure 4.2, we see how the spatial resolution of a schematic stereo setup is influenced by depth. The intersection of a whole pixel of the left and right image forms a quadrilateral and the image plane has a horizontal resolution of $14\,\mathrm{px}$ (=15 rays) in this example. The small blue quadrilateral corresponds to a disparity of $13\,\mathrm{px}$, the orange one to $7\,\mathrm{px}$ and the large yellow one to $3\,\mathrm{px}$. Some parts of each image cannot be seen by the other camera (gray area).

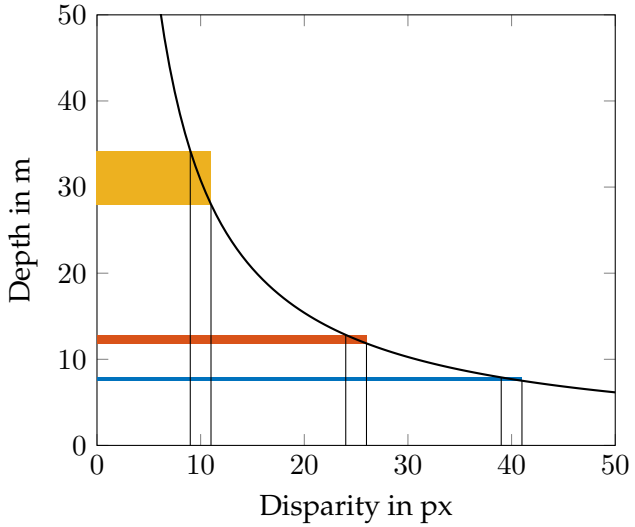


FIGURE 4.1: Corresponding depth of disparity with $f = 1400\,\mathrm{px}$ and $B = 0.22\,\mathrm{m}$. Colored areas are the resulting depth interval of the $\pm 1\,\mathrm{px}$ intervals centered at 10, 25 and $40\,\mathrm{px}$ disparity to visualize the depth differences.

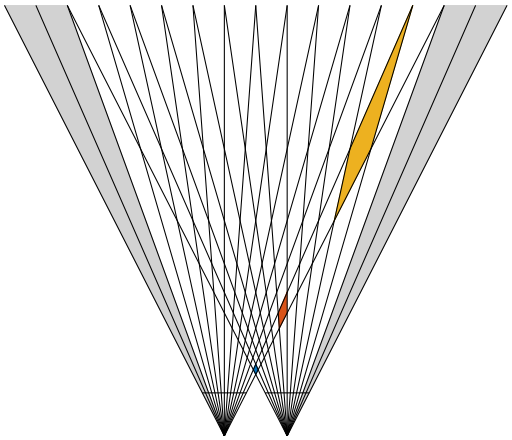


FIGURE 4.2: Schematic top-down view of a stereo camera rig with equally-spaced rays in Euclidean space (meters), which mark pixel borders. Intersection of whole pixels are colored and unmatched areas are gray.

The depth error of a pixel will increase with depth itself. This means that near points are reconstructed relatively accurate, while far points are relatively less accurate because their accuracy is stronger affected by noise. We still want to reconstruct more distant scenery to avoid holes and thus a subsequent loss of immersion. For direct 3D stereo inspection in VR or AR, the depth error fortunately corresponds to human disparity perception (stereopsis). Additionally, the stereo baselines of automotive stereo cameras greatly exceed the mean adult human interpupillary distance of approximately $63\,\mathrm{mm}$ (see Dodgson [Dod04]). Nevertheless, the offset to the tracked HMD and the HMD movement itself potentially increases the observable disparity range, which may requires an accurate reconstruction of even far scenery.

One assumption that we make is that improving the accuracy of the stereo reconstruction will also generally result in a better visualization, *i.e.* fewer artifacts and less noise. To achieve this, we propose to fuse disparity estimates. This is applicable to stereo reconstruction from multiple poses of one camera, *i.e.* temporal pose changes, and also to individual measurements from multiple calibrated stereo cameras for one reconstruction.

This chapter will analyze the stereo matching error and derive two filters for fusing disparity estimates. One is an efficient 1D information filter working only on disparity as the state, and the other one is a full 3D Kalman filter using the u and v coordinates and disparity, which results in more complex computations but potentially higher accuracy for the reconstruction. In the end, we show the advantages of our filter on simulated data, compare both approaches and show limitations of our model. Note that we explicitly cite references for all mathematical formulas and results which we do not derive ourselves.

4.2 Stereo Matching Error Model

Geyer *et al*. [Gey+06] proposed to model the depth error e_z as follows:

$$e_z = \frac{Bf}{d} - \frac{Bf}{d + e_d} = \frac{z^2 e_d}{Bf + z e_d} \tag{4.1}$$

B is the stereo baseline, f the focal length, d the disparity and e_d the disparity or matching error. Thus, to decrease the depth error e_z we have to either increase B or f, decrease e_d or simply move closer to the object for a smaller z. Gallup [Gal11] varied the baseline and focal length (via the resolution) to achieve a desired fixed depth accuracy. Gallup's plane sweep algorithm can choose camera frames of a set of cameras at different times as input, which results in varying baselines. Gallup also downsampled the input images to different resolutions, which allowed the algorithm to choose images with lower focal lengths (in pixels). Geyer *et al*. [Gey+06] increased the stereo baseline and fused measurements from a moving airborne vehicle observing the ground using overlapping images of a mono-camera to reduce the variance of range measurements.

In our scenario with vehicle-mounted cameras, we can only increase the baseline of a front-facing stereo camera up to a certain limit until it would overextend the regular car dimensions, which is not feasible. For a side-facing scenario, one could potentially increase the baseline depending on the vehicle movement similarly to Geyer *et al*. [Gey+06]. This would require accurate pose estimates over longer distances.

Increasing the focal length to decrease the depth error, *i.e*. moving the focal plane farther away from the camera center, would zoom into the scene. This means that we get an even smaller FoV, which is again not feasible for immersive use-cases with HMDS that have a high FoV requirement. Increasing the pixel resolution to increase the focal length (in pixels) is possible, but would require even more computational resources. Furthermore, the resolution is fixed due to the camera hardware. Thus, we can not dynamically increase the accuracy.

Using multiple stereo cameras with overlapping scene views is another option, but it would increase the hardware costs, lower energy efficiency and cause additional overhead to set up and maintain due to camera calibration.

Alternatively, we can fuse disparities temporally between frames with a static stereo baseline, which potentially enables us to decrease the stereo matching error e_d. A front-facing stereo camera scenario enables us to reobserve the scene points more easily than in a side-facing scenario. Reformulating Equation 4.1, we can approximate the maximally acceptable disparity error e_d to achieve a certain desired and fixed depth error e_z:

$$e_d = \frac{e_z B f}{z(z - e_z)} \tag{4.2}$$

e_z does not follow a normal distribution because the disparities are transformed non-linearly from image space to 3D camera space (cf. Sibley *et al*. [SMS07]). This is

especially true for low disparities. Therefore, we can only approximately handle e_z as normally distributed (in contrast to e_d).

Assuming Gaussian noise for the disparity error (with e_d being the standard deviation), the fusion of two uncorrelated observations of the same scene point will result in a lower variance of the disparity error compared to both inputs. If the error is however perfectly correlated between two observations, *e.g.* when running a deterministic stereo matching with unchanged parameters again on the same images, the fusion will not decrease the error. Geyer *et al.* [Gey+06] found in simulated experiments that the error of stereo measurements was correlated up to $60\,\%$ between frames, which suggests that we might underestimate the error with disparity fusion in practice.

The fused disparity estimate $e_{d'}$ is computed from the normalized product of two 1D Gaussian probability density functions with standard deviations e_{d_1} and e_{d_2} (see Barfoot [Bar17]), assuming uncorrelated errors in the stereo matching:

$$\frac{1}{e_{d'}^2} = \frac{1}{e_{d_1}^2} + \frac{1}{e_{d_2}^2} \tag{4.3}$$

The terms above are reciprocals of the variance, which corresponds to the 1D Fisher information matrix, *i.e.* using the canonical parameterization of a normal distribution. The operation sums two information terms to a single information term. To generalize, the fusion of n uncorrelated observations with the same disparity variance results in a new error $e_{d'}$. Thus, we can compute the needed number of observations n with variance e_d^2 to achieve a certain depth error $e_{z'}$ with fusion:

$$\begin{aligned} \frac{1}{e_{d'}^2} &= n \cdot \frac{1}{e_d^2} \\ \Rightarrow n = \frac{e_d^2}{e_{d'}^2} &= \frac{e_d^2}{\left(\frac{e_{z'} B f}{z(z-e_{z'})}\right)^2} = \frac{e_d^2 z^4 + e_d^2 e_{z'}^2 z^2 - 2 e_d^2 e_{z'} z^3}{e_{z'}^2 B^2 f^2} = \mathcal{O}(z^4) \end{aligned} \tag{4.4}$$

We can see that the number of needed frames n is bound by $\mathcal{O}(z^4)$. This means to achieve a target fixed depth error $e_{z'}$ at a depth z, we need quartically more uncorrelated measurements with increasing depth z (assuming $e_{d'} < e_d$).

This seems unfortunate, but for our immersive rendering use case we are again limited by the depth perception of the human eyes. This means that we may not need a fixed depth error in the reconstruction in contrast to other applications, since human perception has an analogous depth error (next to other visual depth cues). Still, doubling either the stereo baseline B or the focal length f reduce the amount of needed images n to a quarter, as is apparent in the denominator of Equation 4.4. Halving the disparity error e_d^2 by *e.g.* using a better stereo matching algorithm achieves the same result. Relaxing the depth accuracy value of the required depth accuracy e_z' to a half has approximately a similar effect.

TABLE 4.1: Examples for the needed uncorrelated observations n of a certain depth z to achieve a desired accuracy $e_{z'} = 0.10\,\mathrm{m}$ with $e_d = 0.5\,\mathrm{px}$, $B = 0.54\,\mathrm{m}$ and $f = 700\,\mathrm{px}$.

Depth	n	**Fused** e_z	**Unfused** e_z
$8.7\,\mathrm{m}$	1	$0.10\,\mathrm{m}$	$0.10\,\mathrm{m}$
$10\,\mathrm{m}$	1.7	$0.10\,\mathrm{m}$	$0.13\,\mathrm{m}$
$20\,\mathrm{m}$	27.7	$0.10\,\mathrm{m}$	$0.53\,\mathrm{m}$
$30\,\mathrm{m}$	140.8	$0.10\,\mathrm{m}$	$1.19\,\mathrm{m}$
$40\,\mathrm{m}$	445.7	$0.10\,\mathrm{m}$	$2.12\,\mathrm{m}$
$50\,\mathrm{m}$	1089.2	$0.10\,\mathrm{m}$	$3.32\,\mathrm{m}$

One example of a temporal fusion situation may be a house facade seen from far away by a sidewards looking camera in a vehicle, so that we always measure the same depth while driving parallel to the building. Table 4.1 shows the needed number of independent observations to get a desired depth accuracy of $e_{z'} = 0.10\,\mathrm{m}$. The intrinsic parameters for this scenario were chosen similarly to the actual parameters of the KITTI visual odometry dataset [GLU12]. Without fusion, the desired accuracy is achieved by points below a depth of ca. $8.7\,\mathrm{m}$. We also compare the fused and unfused depth error in e_z, which is again only approximately normally distributed due to the non-linear transformation from the disparity value.

To achieve the target quality at $50\,\mathrm{m}$, one would need 1090 uncorrelated images. This would mean $109\,\mathrm{s}$ of observing the same scene point at a frame rate of $10\,\mathrm{Hz}$. This is certainly possible in stand-still at a traffic light for ca. $2\,\mathrm{min}$, but will result in correlated errors and thus in the overestimation of the real accuracy. More realistically, it could only be achieved by combining multiple and different observations in a globally shared and updated map. But this would potentially result in a whole other set of multi-faceted challenges.

To summarize, temporal disparity fusion is able to theoretically improve reconstruction results if the Gaussian noise assumption for disparity errors is valid, although many observations may be needed depending on the depth. We further discuss the limitations of this assumption in Subsection 4.6.4.

4.3 1D Extended Information Filter

Static points of a scene will get closer in camera space with each consecutive frame while moving in the direction of the optical axis. For a stereo camera mounted in front of a vehicle, this is always the case while driving forward. A pixel from the last frame warped to the new current frame is now seen from a nearer depth, while its depth accuracy has not changed (even given perfect pose movement). The pixel does not have the accuracy of a new observation at that depth due to the characteristics of the stereo error (see Figure 4.1). For example, a pixel with a depth of $20\,\mathrm{m}$ warped to the new frame may have a new depth of $15\,\mathrm{m}$ in that frame, but their depth accuracy

still remains at the level of a $20\,\mathrm{m}$ observation. Thus, fusing this estimate with a new observation (at *e.g.* $15\,\mathrm{m}$) along that viewing ray in the current frame has to take the different depth accuracies into account, *i.e.* putting more emphasis on the closer and more accurate observation due to the stereo error.

For the fusion of disparity estimates, we derive an extended information filter in 1D. In this way, multiple disparity measurements of the same point can be combined to a single and more accurate disparity estimate. We use a separate filter for every reconstructed point / pixel. Again, we are assuming the stereo matching error is normally distributed, *i.e.* the disparity error but not the depth error. Our state space is in image space and only consists of the disparity as the 1D state variable. Again, we do not use the depth directly but use the disparity in image space for our computations.

A stereo camera moving linearly in 3D corresponds to linear depth changes and thus non-linear disparity changes, which is why we need an *extended* information filter (EIF). We use an information filter here instead of a Kalman filter purely because the fusion equation (see Equation 4.10) is simpler this way and corresponds to the normalized product of two 1D Gaussian probability density functions. Since we are in 1D, operations like matrix inversion are trivial, which is why there is no clear preference between the information or Kalman filter version from a performance perspective.

Having a one dimensional state simplifies the formulation of the extended information filter. The information matrix in 1D is a scalar and the Jacobians represent a 1D derivative. This formulation enables fast computation times as it skips costly matrix operations in higher dimensions such as matrix inversions.

4.3.1 Propagation Step

To propagate the previous state μ_{t-1} with information Ω_{t-1} at time $t-1$ to the new frame at time t, we need to derive the state transition function g. The control variable is the change in depth Δz towards the next frame due to pose movement. We can now derive the state transition function $g(\mu_{t-1}, \Delta z)$ using the disparity formula:

$$\begin{aligned} \bar{z}_t &= z_{t-1} - \Delta z \\ \Rightarrow \frac{Bf_x}{\bar{\mu}_t} &= \frac{Bf_x}{\mu_{t-1}} - \Delta z \\ \Leftrightarrow \bar{\mu}_t &= \frac{\mu_{t-1}}{1 - \frac{\Delta z \mu_{t-1}}{Bf_x}} = g(\mu_{t-1}, \Delta z) \end{aligned} \tag{4.5}$$

The bar denotes the propagated version of the old measurement to the new frame as we have not integrated the new measurement yet. B stands for the stereo baseline in meters and f_x for the horizontal focal length in pixels. The vehicle movement and camera frame rate determine the size of Δz. In practice, we implicitly compute Δz by warping the pixel to the new frame, but we still need explicitly compute it to

propagate the information. Now, the derivative of the state transition function is:

$$g'(\mu_{t-1}, \Delta z) = \frac{\partial g(\mu_{t-1}, \Delta z)}{\partial \mu_{t-1}} = \left(\frac{\bar{\mu}_t}{\mu_{t-1}}\right)^2 = J_{\bar{\mu}_t} \tag{4.6}$$

Then, we propagate the information Ω_{t-1} to $\bar{\Omega}_t$. For that, we propagate the previous information Ω_{t-1} with the Jacobian $J_{\bar{\mu}_t}$, which in 1D is just the derivative of the state transition function, *i.e.* a scalar factor. Additionally, we can propagate a state variance term σ_z^2 along the optical axis (similar to Schöps *et al.* [Sch+15]) to model uncertainty in pose estimation in 1D. Again, we need to derive the Jacobian $J_{\Delta z}$, which is the derivative of the state transition function with respect to Δz:

$$J_{\Delta z} = \frac{\partial g(\mu_{t-1}, \Delta z)}{\partial \Delta z} = \frac{(\bar{\mu}_t)^2}{B f_x} \tag{4.7}$$

To conclude the propagation step, we write the full propagation equation as follows:

$$\bar{\Omega}_t = \left(J_{\bar{\mu}_t} \Omega_{t-1}^{-1} J_{\bar{\mu}_t} + J_{\Delta z} \sigma_z^2 J_{\Delta z}\right)^{-1} = \left(\left(\frac{\bar{\mu}_t}{\mu_{t-1}}\right)^4 \Omega_{t-1}^{-1} + \frac{(\bar{\mu}_t)^4}{(B f_x)^2} \sigma_z^2\right)^{-1} \tag{4.8}$$

Both resulting terms in the previous equation consist of a scalar multiplied by a variance (= inverse of information). The factor for the left term is greater than 1 in most cases because we are moving forwards into the scene: With a propagated disparity of $\bar{\mu}_t = 20\,\text{px}$ and a previous disparity of $\mu_{t-1} = 21\,\text{px}$, we can compute the factor $\left(\frac{21}{20}\right)^4 \approxeq 1.22$. So, the existing variance will grow by a factor of 1.22. The factor for the right term punishes higher propagated disparity values (= near points) harder than lower disparities: With $B = 0.54\,\text{px}$ and $f_x = 700\,\text{px}$, the factor for $\bar{\mu}_t = 20\,\text{px}$ is $\frac{20^4}{(0.54 \cdot 700)^2} \approxeq 1.12$ and for $\bar{\mu}_t = 21\,\text{px}$ is $\frac{21^4}{(0.54 \cdot 700)^2} \approxeq 1.36$. A static pose error along the optical axis has a greater relative influence on near and very accurate propagated depth observations. This makes sense because a static pose error now contributes more to the overall error than a possibly very accurate short-range stereo matching observation.

4.3.2 Update Step

We have to use the propagated information and the information of new measurement to finally compute the new, current state estimate μ_t in the update step. Since the measurement space is the state space, we can simplify the update equations by omitting the observation matrix H_t. The measurement error in disparity is modeled as a normal distribution with variance e_d^2. Thus, to fuse the propagated information $\bar{\Omega}_t$ and the information of the new measurement $\frac{1}{e_d^2}$ to the current information estimate Ω_t , we can simply compute their sum (see also Equation 4.3):

$$\Omega_t = \bar{\Omega}_t + \frac{1}{e_d^2} \tag{4.9}$$

To compute the new state μ_t with the new disparity estimate d_t, we can write:

$$\mu_t = \frac{\left(\bar{\Omega}_t \bar{\mu}_t + \frac{1}{e_d^2} d_t\right)}{\Omega_t} \tag{4.10}$$

This is a weighted average. Both disparity estimates $\bar{\mu}_t$ and d_t have a weight corresponding to the magnitude of their information and are then normalized by the information sum Ω_t, *i.e.* the sum of their weights.

4.4 3D Extended Kalman Filter

Our 1D EIF uses the disparity as the only state variable, which proves to be effective and efficient since the disparity generally provides the greatest source of variance in stereo construction. Still, the u and v coordinates in image space are affected by the quality of the camera calibration, which manifests as angular or pointing errors (as proposed by Murray and Little [ML04]). A more complete model, which we will derive in this section, would thus take this into account. For stereo quantization noise, Blostein and Huang [BH87] assume that a point's actual position is uniformly distributed inside the 3D volume of a pixel, which would correspond to a trapezoid from a top-down view. We do not make this assumption because we assume sub-pixel disparity accuracy, which pinpoints the correct location with some uncertainty.

Assuming Gaussian noise, the covariance of a point in image space ($[u, v, d]^T$) has the form of a Gaussian ellipsoid. From a top-down view in image space, it is a 2D ellipse. When propagating the covariance to the new position in a new frame, the depth (or disparity d) of a point will also affect the u and v image space coordinates. Thus, movement induces correlations between the image space coordinates. This means that the non-diagonal elements of the covariance matrix are not zero, making corresponding ellipses appear rotated or tilted. With greater angular differences from the optical axis, the covariance will shift more away from the disparity axis and more towards the u and v axis (assuming that the disparity is responsible for the greatest variance). An exception is the case when the point does lie on the optical axis, because that would mean that the movement does not cause correlation in such a way.

Now, given such a good camera calibration that the u and v error is significantly smaller than the disparity error, the ellipses are highly eccentric. These "narrow" ellipses are mainly defined by their variance along the disparity axis. Intuitively, fusing very differently oriented ellipses may result in a better state estimate, since the intersection of their covariances may be only a small volume. Thus, despite a very erroneous stereo matching, the same corresponding point can now potentially be accurately triangulated over multiple frames alone given a very precise pose estimate. In practice however, an erroneous depth would make it harder to find the correct

correspondences between frames. Again, the point seen from camera space cannot lie on the optical axis in this case since their covariance is not a tilted ellipsoid.

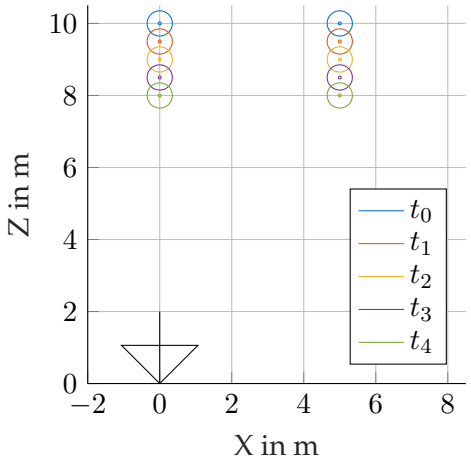
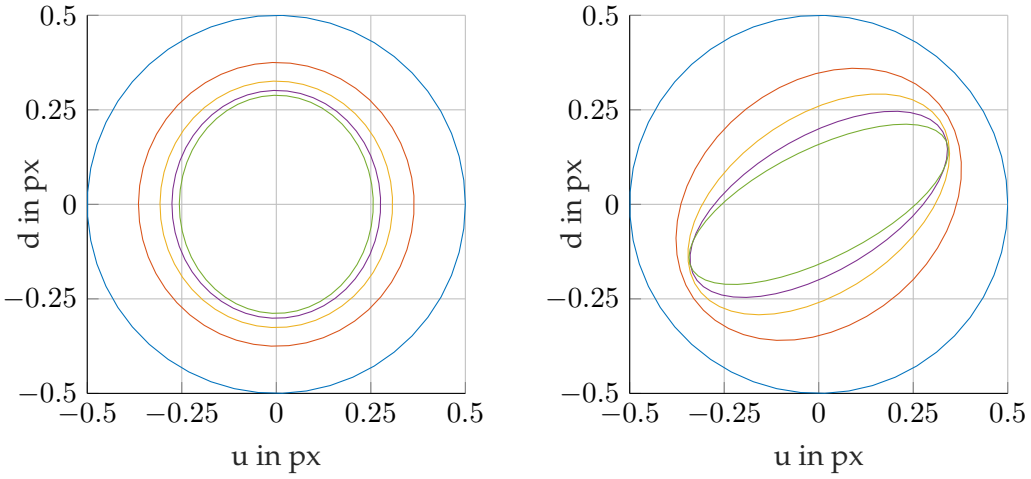


FIGURE 4.3: Top-down view of the camera space with relative point measurements over multiple frames. One point lies along the optical axis (left set) and the other one is shifted (right set).

For demonstration purposes, let us consider a 2D example of the proposed 3D filtering algorithm, which we will derive in the following. Figure 4.3 shows the left camera coordinate system of a stereo camera with two points seen at multiple time steps while the stereo camera is moving forward along the Z axis. The camera moves $0.5\,\mathrm{m}$ each frame and has the following relevant intrinsics: $B = 0.54\,\mathrm{m}$, $f_x = 700\,\mathrm{px}$ and $c_x = 500\,\mathrm{px}$. At t_0, the left point is seen at $[0, 10]^T$ in camera space, which lies on the optical axis, and the right point is seen at $[5, 10]^T$, which is shifted from the optical axis.



(a) Measurements along the optical axis (left point).

(b) Shifted measurements (right point).

FIGURE 4.4: Comparing the covariance of fused measurements with the 3D EKF over multiple frames as shown in Figure 4.3 with corresponding colors. Ellipses correspond to the $1 \cdot \sigma$ contour.

Figure 4.4 compares the covariance during the temporal fusion in image space

with the same starting variance of $0.5^2\,\mathrm{px}^2$ for the horizontal pixel position u and disparity d with negligible process noise. The final (green) ellipse of the shifted point in Figure 4.4b is tilted as described previously. The final ellipse of the point along the optical axis in Figure 4.4a is not tilted since the starting variance for u and d is the same. Consequently, the disparity error of a point after fusion is smaller for the shifted point on the right than the point on the optical axis. The 1D EIF only produces fused disparity estimates that correspond to the point along the optical axis. So, using two extra dimensions has the potential to lower the disparity error even further. We will compare both filters in Subsection 4.5.2 to verify this claim.

4.4.1 Propagation Step

We explained the basic algorithm of an EKF in Section 2.4 and here we detail the derivation of our 3D model. First of all, the state space is the 3D image space coordinate $\boldsymbol{\mu} = [u, v, d]^T$ and the covariance matrix $\boldsymbol{\Sigma}$ is now a 3×3 matrix instead of a scalar. Furthermore, since our state space is equal to the stereo measurement space, we can again simplify the EKF equations by setting $\mathbf{H} = \mathbf{I^3}$. We restate the EKF equations in the following for clarity.

To propagate the previous state $\boldsymbol{\mu_{t-1}}$ and covariance $\boldsymbol{\Sigma_{t-1}}$, we need to define the state transition function g, its Jacobian $\mathbf{G}$ and the covariance of the process noise $\mathbf{R_t}$:

$$\bar{\boldsymbol{\mu}}_\mathbf{t} = g(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \tag{4.11}$$

$$\bar{\boldsymbol{\Sigma}}_\mathbf{t} = \mathbf{G_t}\boldsymbol{\Sigma_{t-1}}\mathbf{G_t^T} + \mathbf{R_t} \tag{4.12}$$

We will define and derive all propagation components in the following.

State Transition Function g

First, we define the state $\boldsymbol{\mu}$ and the control variable $\mathbf{c}$ at time t as follows:

$$\boldsymbol{\mu} = \begin{bmatrix} u \\ v \\ d \end{bmatrix} \quad \mathbf{c_t} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{4.13}$$

u is the horizontal and v the vertical pixel coordinate in image space. d is the disparity. Δx, Δy and Δz is the position change along the corresponding 3D axis in 3D camera space (X, Y, Z). With this, g can be divided into three separate functions for each state dimension:

$$g(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) = \begin{bmatrix} g_1(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \\ g_2(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \\ g_3(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \end{bmatrix} \tag{4.14}$$

We derive each transition function separately. We start with g_3 by considering that the points are always moving closer in camera space ($-\Delta z \leq 0$) given forwards motion in our front camera vehicle setup:

$$\begin{aligned}
& z_t = z_{t-1} - \Delta z \\
\Leftrightarrow \quad & \frac{Bf_x}{d_t} = \frac{Bf_x}{d_{t-1}} - \Delta z \\
\Leftrightarrow \quad & Bf_x = \left(\frac{Bf_x}{d_{t-1}} - \Delta z\right) d_t \\
\Leftrightarrow \quad & \frac{Bf_x}{\frac{Bf_x}{d_{t-1}} - \Delta z} = d_t \\
\Leftrightarrow \quad & \frac{Bf_x d_{t-1}}{Bf_x - d_{t-1}\Delta z} = d_t \\
\Leftrightarrow \quad & \frac{d_{t-1}}{1 - \frac{d_{t-1}\Delta z}{Bf_x}} = d_t = g_3(\mathbf{c_t}, \boldsymbol{\mu_{t-1}})
\end{aligned} \tag{4.15}$$

Next, for g_1 and g_2, we use the perspective projection equations of a point in camera space at $[x, y, z]^T$ to transform the point to image space:

$$\begin{aligned}
x &= \frac{u - c_x}{f_x} z & y &= \frac{v - c_y}{f_y} z \\
&= \frac{u - c_x}{f_x} \frac{Bf_x}{d} & &= \frac{v - c_y}{f_y} \frac{Bf_x}{d}
\end{aligned}$$

Now, we can derive g_1 as follows using the change in 3D coordinates in camera space and g_3:

$$\begin{aligned}
& x_t = x_{t-1} + \Delta x \\
\Leftrightarrow \quad & \frac{(u_t - c_x)}{f_x} \frac{Bf_x}{d_t} = \frac{(u_{t-1} - c_x)}{f_x} \frac{Bf_x}{d_{t-1}} + \Delta x \\
\Leftrightarrow \quad & u_t = \left(\frac{(u_{t-1} - c_x)}{f_x} \frac{Bf_x}{d_{t-1}} + \Delta x\right) \frac{f_x d_t}{Bf_x} + c_x \\
\Leftrightarrow \quad & u_t = \frac{(u_{t-1} - c_x)\, d_t}{d_{t-1}} + \frac{\Delta x d_t}{B} + c_x \\
\Leftrightarrow \quad & u_t = d_t \left(\frac{u_{t-1} - c_x}{d_{t-1}} + \frac{\Delta x}{B}\right) + c_x \\
\xLeftrightarrow{\text{Eq. 4.15}} \quad & u_t = \frac{d_{t-1}}{1 - \frac{d_{t-1}\Delta z}{Bf_x}} \left(\frac{u_{t-1} - c_x}{d_{t-1}} + \frac{\Delta x}{B}\right) + c_x = g_1(\mathbf{c_t}, \boldsymbol{\mu_{t-1}})
\end{aligned} \tag{4.16}$$

g_2 can be derived analogously to g_1 by assuming $f_x = f_y$ and using the following mapping: $u \to v$, $x \to y$, $\Delta x \to \Delta y$ and $c_x \to c_y$. This gives us:

$$v_t = \frac{d_{t-1}}{1 - \frac{d_{t-1}\Delta z}{Bf_x}} \left(\frac{v_{t-1} - c_y}{d_{t-1}} + \frac{\Delta y}{B}\right) + c_y = g_2(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \tag{4.17}$$

To summarize, g has the following form:

$$g(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) = \begin{bmatrix} g_1(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \\ g_2(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \\ g_3(\mathbf{c_t}, \boldsymbol{\mu_{t-1}}) \end{bmatrix} = \begin{bmatrix} \frac{d_{t-1}}{1-\frac{d_{t-1}\Delta z}{Bf_x}} \left(\frac{u_{t-1}-c_x}{d_{t-1}} + \frac{\Delta x}{B} \right) + c_x \\ \frac{d_{t-1}}{1-\frac{d_{t-1}\Delta z}{Bf_x}} \left(\frac{v_{t-1}-c_y}{d_{t-1}} + \frac{\Delta y}{B} \right) + c_y \\ \frac{d_{t-1}}{1-\frac{d_{t-1}\Delta z}{Bf_x}} \end{bmatrix} \tag{4.18}$$

$$= \frac{d_{t-1}}{1-\frac{d_{t-1}\Delta z}{Bf_x}} \begin{bmatrix} \left(\frac{u_{t-1}-c_x}{d_{t-1}} + \frac{\Delta x}{B} \right) \\ \left(\frac{v_{t-1}-c_y}{d_{t-1}} + \frac{\Delta y}{B} \right) \\ 1 \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \\ 0 \end{bmatrix} \tag{4.19}$$

State Transition Jacobian $\mathbf{G_t}$

Now that we have g, computing the partial derivatives for $\mathbf{G_t}$ is straightforward:

$$\mathbf{G_t} = \begin{bmatrix} \frac{\partial g_1}{\partial u} & \frac{\partial g_1}{\partial v} & \frac{\partial g_1}{\partial d} \\ \frac{\partial g_2}{\partial u} & \frac{\partial g_2}{\partial v} & \frac{\partial g_2}{\partial d} \\ \frac{\partial g_3}{\partial u} & \frac{\partial g_3}{\partial v} & \frac{\partial g_3}{\partial d} \end{bmatrix} = \begin{bmatrix} \frac{Bf_x}{Bf_x - d_{t-1}\Delta z} & 0 & \frac{Bf_x(\Delta z(u_{t-1}-c_x)+f_x\Delta x)}{(Bf_x-d_{t-1}\Delta z)^2} \\ 0 & \frac{Bf_x}{Bf_x-d_{t-1}\Delta z} & \frac{Bf_x(\Delta z(v_{t-1}-c_y)+f_x\Delta y)}{(Bf_x-d_{t-1}\Delta z)^2} \\ 0 & 0 & \left(\frac{d_t}{d_{t-1}}\right)^2 \end{bmatrix} \tag{4.20}$$

The Jacobian $\mathbf{G_t}$ has empty entries because the differentiating variable is not always present in the corresponding formula, *e.g.* horizontal pixel movement does not affect vertical pixel movement.

Using an intermediate result of the derivation of g_3 in Equation 4.15, we can define a shorthand s_1:

$$d_t = \frac{Bf_x d_{t-1}}{Bf_x - d_{t-1}\Delta z} \tag{4.21}$$

$$s_1 := \frac{\partial g_1}{\partial u} = \frac{\partial g_2}{\partial v} = \frac{Bf_x}{Bf_x - d_{t-1}\Delta z} = \frac{d_t}{d_{t-1}} \tag{4.22}$$

This leads us to a more compact form for $\mathbf{G_t}$:

$$\mathbf{G_t} = \begin{bmatrix} s_1 & 0 & s_1^2 \frac{(\Delta z(u_{t-1}-c_x)+f\Delta x)}{Bf} \\ 0 & s_1 & s_1^2 \frac{(\Delta z(v_{t-1}-c_y)+f\Delta y)}{Bf} \\ 0 & 0 & s_1^2 \end{bmatrix} \tag{4.23}$$

Process Noise Covariance $\mathbf{R_t}$

We model the covariance of the process noise $\mathbf{R_t}$ with two components: $\mathbf{T}$ represents the translational relative pose error in meters and $\mathbf{U}$ the rotational relative pose error

in radians. Each degree of freedom of the pose can be modelled individually with independent Gaussian noise:

$$\mathbf{T} = \begin{bmatrix} \sigma_{t_x}^2 & 0 & 0 \\ 0 & \sigma_{t_y}^2 & 0 \\ 0 & 0 & \sigma_{t_z}^2 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} \sigma_{r_x}^2 & 0 & 0 \\ 0 & \sigma_{r_y}^2 & 0 \\ 0 & 0 & \sigma_{r_z}^2 \end{bmatrix} \tag{4.24}$$

Since our state space is the image space, we need to propagate the translational error from meters to pixels with a Jacobian $\mathbf{J_{M_t}}$ at time t. Similarly, we need to propagate the rotational error first to meters with a Jacobian $\mathbf{J_{R_t}}$ at time t, which we then propagate to pixels with $\mathbf{J_{M_t}}$. Both of these error components constitute the final process covariance $\mathbf{R_t}$ at time t:

$$\mathbf{R_t} = \mathbf{J_{M_t}} \mathbf{T} \mathbf{J_{M_t}^T} + \mathbf{J_{M_t}} \mathbf{J_{R_t}} \mathbf{U} \mathbf{J_{R_t}^T} \mathbf{J_{M_t}^T} = \mathbf{J_{M_t}} \left(\mathbf{T} + \mathbf{J_{R_t}} \mathbf{U} \mathbf{J_{R_t}^T} \right) \mathbf{J_{M_t}^T} \tag{4.25}$$

Meter-to-pixel Jacobian $\mathbf{J_{M_t}}$

We can derive $\mathbf{J_{M_t}}$ analogously to $\mathbf{G_t}$:

$$\mathbf{J_{M_t}} = \begin{bmatrix} \frac{\partial g_1}{\partial \Delta x} & \frac{\partial g_1}{\partial \Delta y} & \frac{\partial g_1}{\partial \Delta z} \\ \frac{\partial g_2}{\partial \Delta x} & \frac{\partial g_2}{\partial \Delta y} & \frac{\partial g_2}{\partial \Delta z} \\ \frac{\partial g_3}{\partial \Delta x} & \frac{\partial g_3}{\partial \Delta y} & \frac{\partial g_3}{\partial \Delta z} \end{bmatrix} = \begin{bmatrix} \frac{f_x d_{t-1}}{B f_x - d_{t-1} \Delta z} & 0 & \frac{f_x d_{t-1} (B(u_{t-1} - c_x) + d_{t-1} \Delta x)}{(B f_x - d_{t-1} \Delta z)^2} \\ 0 & \frac{f_x d_{t-1}}{B f_x - d_{t-1} \Delta z} & \frac{f_x d_{t-1} (B(v_{t-1} - c_y) + d_{t-1} \Delta y)}{(B f_x - d_{t-1} \Delta z)^2} \\ 0 & 0 & \frac{d_t^2}{B f_x} \end{bmatrix} \tag{4.26}$$

Again, we can define a shorthand (s_2) and use d_t to write $\mathbf{J_{M_t}}$ in a more compact form:

$$s_2 := \frac{\partial g_1}{\partial \Delta x} = \frac{\partial g_2}{\partial \Delta y} = \frac{f_x d_{t-1}}{B f_x - d_{t-1} \Delta z} = \frac{d_t}{B} \tag{4.27}$$

$$\mathbf{J_{M_t}} = \begin{bmatrix} s_2 & 0 & s_2^2 \frac{(B(u_{t-1} - c_x) + d_{t-1} \Delta x)}{d_{t-1} f_x} \\ 0 & s_2 & s_2^2 \frac{(B(v_{t-1} - c_y) + d_{t-1} \Delta y)}{d_{t-1} f_x} \\ 0 & 0 & \frac{d_t^2}{B f_x} \end{bmatrix} \tag{4.28}$$

Radian-to-meter Jacobian $\mathbf{J_{R_t}}$

To model the relative rotational error in radians, we use extrinsic Euler angles with the XYZ convention. This corresponds to rotations around a fixed global axis in the following sequence: X, then Y, then Z; corresponding to pitch, roll and yaw. Here ΔX, ΔY and ΔZ are the relative rotation Euler angles used in the transformation from the last frame to the current frame. Our final rotation matrix $\mathbf{R_{xyz}}$ to transform a 3D point $\mathbf{p}$ can be built via the three rotation matrices $\mathbf{R_x}$, $\mathbf{R_y}$ and $\mathbf{R_z}$, which each represent a rotation around their particular global axis. The idea is to transform the

world coordinate system before we apply a new rotation and restore the previous state after the new rotation. As a result, the rotation sequence is flipped compared to local rotations. Here, we are using shorthand notations for the sine and cosine terms exclusively, *e.g.* $s_1 = \sin(\Delta X)$ or $c_3 = \cos(\Delta Z)$:

$$\mathbf{R_{xyz}p} = \mathbf{R_x R_y R_z R_y^T R_x^T (R_x R_y R_x^T (R_x))p} \tag{4.29}$$

$$= \mathbf{R_x R_y R_z p} \tag{4.30}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & -s_1 \\ 0 & s_1 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4.31}$$

$$= \begin{bmatrix} zs_2 + c_2(xc_3 - ys_3) \\ -zc_2s_1 + s_1s_2(xc_3 - ys_3) + c_1(yc_3 + xs_3) \\ zc_1c_2 + s_1(yc_3 + xs_3) - c_1s_2(xc_3 - ys_3) \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \tag{4.32}$$

We are assuming independent Gaussian noise for each Euler angle change ΔX, ΔY and ΔZ. Thus, every component is actually the combination of the true Euler angle change $[\Delta X_{true}, \Delta Y_{true}, \Delta Z_{true}]^T$ and a normally distributed errors for each angle term $[e_x, e_y, e_z]^T$:

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} = \begin{bmatrix} \Delta X_{true} \\ \Delta Y_{true} \\ \Delta Z_{true} \end{bmatrix} + \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \tag{4.33}$$

With this assumption, we can compute the Jacobian $\mathbf{J_{R_t}}$. Here, we are again using the shorthand terms for sine and cosine:

$$\mathbf{J_{R_t}} = \begin{bmatrix} \frac{\partial h_1}{\partial \Delta X} & \frac{\partial h_1}{\partial \Delta Y} & \frac{\partial h_1}{\partial \Delta Z} \\ \frac{\partial h_2}{\partial \Delta X} & \frac{\partial h_2}{\partial \Delta Y} & \frac{\partial h_2}{\partial \Delta Z} \\ \frac{\partial h_3}{\partial \Delta X} & \frac{\partial h_3}{\partial \Delta Y} & \frac{\partial h_3}{\partial \Delta Z} \end{bmatrix} \tag{4.34}$$

$$= \begin{bmatrix} 0 & s_2(ys_3 - xc_3) + zc_2 & -c_2(xs_3 + yc_3) \\ c_1s_2(xc_3 - ys_3) - s_1(xs_3 + yc_3) - zc_1c_2 & s_1(c_2(xc_3 - ys_3) + zs_2) & c_1(xc_3 - ys_3) - s_1s_2(xs_3 + yc_3) \\ s_1s_2(xc_3 - ys_3) + c_1(xs_3 + yc_3) - zs_1c_2 & -c_1(c_2(xc_3 - ys_3) + zs_2) & c_3(xs_1 + yc_1s_2) + s_3(xc_1s_2 - ys_1) \end{bmatrix} \tag{4.35}$$

4.4.2 Update Step

With the propagated state $\bar{\boldsymbol{\mu}}_\mathbf{t}$ and $\bar{\boldsymbol{\Sigma}}_\mathbf{t}$, we can compute the Kalman gain $\mathbf{K_t}$ and the new state $\boldsymbol{\mu}_\mathbf{t}$ with covariance $\boldsymbol{\Sigma}_\mathbf{t}$:

$$\mathbf{K_t} = \bar{\boldsymbol{\Sigma}}_\mathbf{t}(\bar{\boldsymbol{\Sigma}}_\mathbf{t} + \mathbf{Q_t})^{-1} \tag{4.36}$$

$$\boldsymbol{\mu}_\mathbf{t} = \bar{\boldsymbol{\mu}}_\mathbf{t} + \mathbf{K_t}(\mathbf{z_t} - \bar{\boldsymbol{\mu}}_\mathbf{t}) \tag{4.37}$$

$$\boldsymbol{\Sigma}_\mathbf{t} = (\mathbf{I} - \mathbf{K_t})\bar{\boldsymbol{\Sigma}}_\mathbf{t} \tag{4.38}$$

The measurement covariance matrix $\mathbf{Q_t}$ is in image space. We can define the variance for the u and v pointing error and the stereo matching error for the disparity d directly as proposed by Murray and Little [ML04]:

$$\mathbf{Q} = \begin{bmatrix} \sigma_u^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_d^2 \end{bmatrix} \tag{4.39}$$

Note that we assume here that $\mathbf{Q} = \mathbf{Q_t}$, which means that all measurements have the same covariance in image space. With extra input from stereo matching, $\mathbf{Q_t}$ can be different for each measurement due to a dynamic σ_d^2. We discuss this further in Subsection 4.6.3.

4.4.3 Fusion of Two Filters

Usually, most applications will only use a single Kalman filter to represent a single entity like the state of a robot. Here, we model each point of our dense reconstruction, including new measurements, with a Kalman filter. We use input images with a resolution in the order of a million pixels, which are all individually modeled by a Kalman (or information) filter. After warping the previous pixels of an iteration to the current frame, we now have multiple estimates of the 3D position and uncertainty of a particular pixel, since a warped point can occupy the same pixel as a new observation from the stereo camera. These estimates can either be from the same scene point or from a completely different scene point (due to the camera movement). If they correspond to the same scene point according to some criteria, the estimates can be fused: The warped pixel is described as the propagated Kalman filter state $\bar{\boldsymbol{\mu}}_\mathbf{t}$ and the new estimate of the disparity map is the observation input $\mathbf{z_t}$ for the Kalman filter, which can be combined according to the update step. Figure 4.5a illustrates this scenario inside a pixel bin.

(a) One filter case. (b) Two filter case.

FIGURE 4.5: Standard Kalman filter update case (left) and two filter case (right) of propagated Kalman filter states $\bar{\boldsymbol{\mu}}_\mathbf{t}$ and new measurement $\mathbf{z_t}$ at time t inside a pixel bin.

Nonetheless, due to the probabilistic nature of the stereo estimates, it can happen that over time two propagated Kalman filter states land inside the same pixel bin and

belong to the same scene point. For example, the first propagated state $\bar{\boldsymbol{\mu}}_{\mathbf{t_1}}$ could be located more towards the left tail of the probability distribution function of the disparity error, while the second propagated state $\bar{\boldsymbol{\mu}}_{\mathbf{t_2}}$ is located more towards the right tail. A third observation in the middle could cascade into the combination of all estimates to a single one, *i.e.* we also have to fuse the two Kalman filters, which we will describe in the following. Figure 4.5b shows this scenario.

A Kalman filter expects a new measurement to correct its propagated state. When we decide that the state estimates of two filters inside a pixel bin belong to the same scene point, we want to combine both filters to a single instance. For this, we can simply input one of the two filters as a new measurement to the other one. Given two EKFs with propagated states $\bar{\boldsymbol{\mu}}_{\mathbf{t_1}}$ and $\bar{\boldsymbol{\mu}}_{\mathbf{t_2}}$ with covariances $\bar{\boldsymbol{\Sigma}}_{\mathbf{t_1}}$ and $\bar{\boldsymbol{\Sigma}}_{\mathbf{t_2}}$, we can substitute:

$$\bar{\boldsymbol{\mu}}_{\mathbf{t}} = \bar{\boldsymbol{\mu}}_{\mathbf{t_1}} \qquad \bar{\boldsymbol{\Sigma}}_{\mathbf{t}} = \bar{\boldsymbol{\Sigma}}_{\mathbf{t_1}} \tag{4.40}$$

$$\mathbf{z}_{\mathbf{t}} = \bar{\boldsymbol{\mu}}_{\mathbf{t_2}} \qquad \mathbf{Q}_{\mathbf{t}} = \bar{\boldsymbol{\Sigma}}_{\mathbf{t_2}} \tag{4.41}$$

We set the second EKF to be a new measurement with state $\mathbf{z}_{\mathbf{t}}$ and covariance $\mathbf{Q}_{\mathbf{t}}$ as input to the first EKF with propagated state $\bar{\boldsymbol{\mu}}_{\mathbf{t}}$ and covariance $\bar{\boldsymbol{\Sigma}}_{\mathbf{t}}$. In short, the second EKF is now the observation input of the first EKF. This allows us to fuse two EKFs. The result is a single combined filter with lower uncertainty compared to the individual inputs.

This approach is agnostic on whether one of the filters is a new observation or not, which is why we can handle any fusion situations that may be induced by camera movement or pixel sampling. We use this approach for the 3D EKF as well as the 1D EIF.

4.5 Filter Evaluation on Simulated Data

In this section, we simulate stereo measurements to compare different fusion approaches. Simulation allows us to directly compare fusion approaches outside their frameworks and thus enables us to exclude other influences. For example, in a real application, we would first have to find plausible point correspondences between frames before we fuse points. *What* we fuse matters as well as *how* we fuse. We cover the *what* in Chapter 5.

4.5.1 1D Fusion Simulation

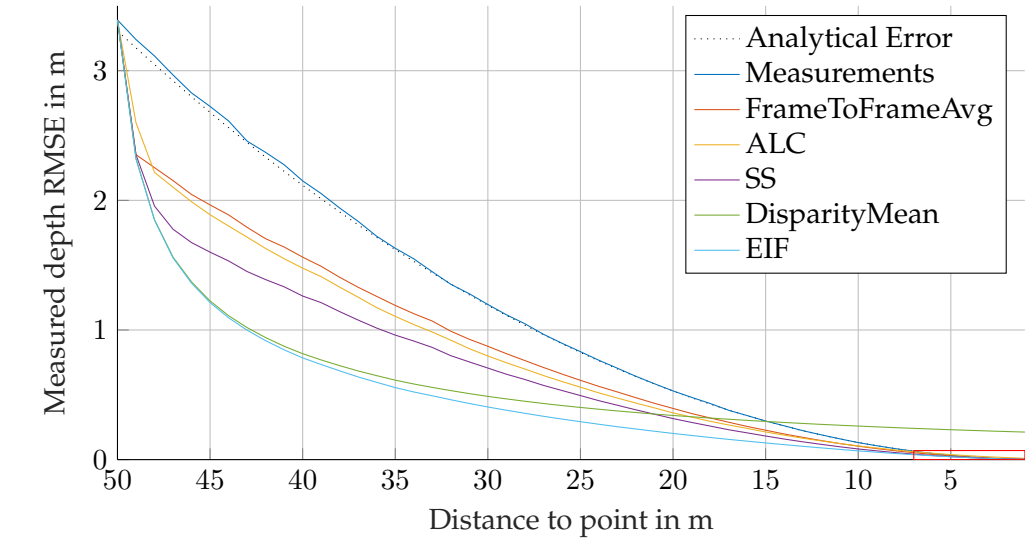
To compare our 1D disparity fusion to other proposed approaches as well as naive approaches, we simulate 1D stereo measurements along the optical axis of the left camera. We move towards the scene point with our stereo camera and simulate matching errors by applying Gaussian noise to the true disparity value using Octave [Eat+16]. We use a standard deviation of $e_d = 0.5\,\mathrm{px}$ for the matching error (cf. [HNS19a] and Figure 4.11 for empirical stereo error distributions). Furthermore, we

use parameters similar to the KITTI dataset by choosing $B = 0.54\,\mathrm{m}$, $f = 700\,\mathrm{px}$, a speed of $36\,\mathrm{km\,h^{-1}}$ and a sampling rate of $10\,\mathrm{Hz}$. We start our measurement from a distance of $50\,\mathrm{m}$ and drive closer to the point, which lies directly in front of the left (main) camera. One trial consists of fusing all measurements from start to end.

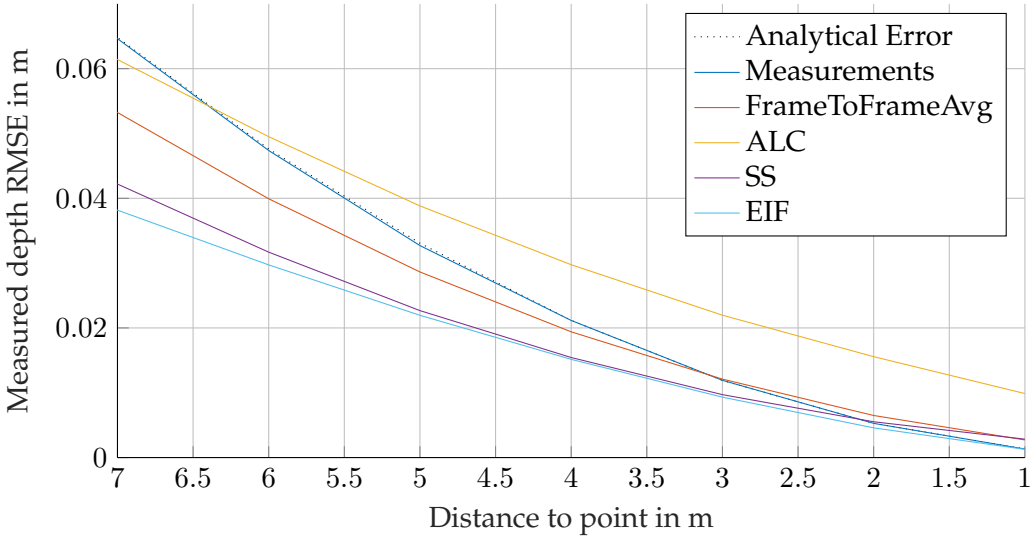
Here, we will describe the used fusion approaches in the following and highlight their names in bold. We will also evaluate the raw measurements as a baseline named **Measurements**, accompanied with the expected error of the stereo matching error model from Section 4.2 as **Analytical Error**. For the fusion, the **SS** approach (inspired by StereoScan [GZS11]) uses two and **ALC** (inspired by Alcantarilla *et al.* [ABD13]) three consecutive frames. **DisparityMean** and **FrameToFrameAvg** both incorporate the whole history of measurements. In contrast to **DisparityMean**, **FrameToFrameAvg** weights the accumulated history and the new measurement equally. **DisparityMean** is the mean of all measured disparities, which means that every measurement is weighted equally along the whole sequence. **EIF** is our 1D EIF approach.

Figure 4.6 shows the mean root mean square error (mean RMSE) of $25\,000$ Monte Carlo trials with all fusion approaches, as well as the mean raw, unfused measurements as the error baseline and the expected analytical error computed via Equation 4.1. Our **EIF** approach incorporates every measurement and outperforms all other fusion approaches. Clearly, fusing measurements helps reduce the depth error. At close ranges, there is a point where the result of the other fusion approaches in the front-facing scenario is worse than the accuracy of the new, raw measurement (see Figure 4.6b). At that point, these fusion approaches will have a detrimental effect on reconstruction quality, while our approach is able to weigh the previously fused measurements appropriately even at close ranges. This has practical consequences as can be seen in Pire *et al.* [Pir+18]: Their approach improves the aggregated median depth error per depth on KITTI VO for far depths compared to ELAS, but is outperformed by the unfused, raw stereo below ca. $13\,\mathrm{m}$. Also, changing parameters like the movement speed and frames per second of the camera will change performance rankings of other approaches since they rely differently on past frames. For example, bigger depth jumps via a higher velocity may favor approaches which put more emphasis on the new measurement than the fused history. Our approach is not affected by the camera sampling in this way and still outperforms the others.

Sibley *et al.* [SMS07] showed that averaging of Gaussian disparity estimates in Euclidean coordinates (*i.e.* as depths) results in a long range bias due to the nonlinear transformation, which results in a skewed Gaussian distribution. We did not account for this effect for the fusion or the computation of mean results. The effect was in the order of centimeters at a depth of $50\,\mathrm{m}$, which corresponded to ca. one percent of the error. This can explain why the **Analytical Error** curve is slightly off the **Measurements** in Figure 4.6 for higher depths.



(a) Full range. Close up box highlighted in red (bottom right).



(b) Close up. DisparityMean is not visible in this scope.

FIGURE 4.6: Average RMSE of 25 000 Monte Carlo trials of different disparity/depth fusion methods. One trial consists of one meter steps from 50 m to 1 m.

4.5.2 3D Filter Comparison

The main motivation behind the 3D EKF was to use additional dimensions to better triangulate 3D scene points. Now, we hypothesize that the 1D EIF and 3D EKF behave very similarly for points along the optical axis, since the viewing angle of a 3D point remains static throughout constant camera movement along the optical axis. Here, we investigate how both filters compare against each other with 3D points that may not lie on the optical axis (see Figure 4.3). We again use a Monte Carlo simulation to model stereo camera movement along the optical axis. We observe a 3D scene point from a depth of 50 m to 5 m in 10 steps (including start and end). We offset the relative start position of the observed point in both horizontal (X) and vertical (Y)

directions from the optical axis by up to $10\,\mathrm{m}$ to investigate off-optical-axis effects. These positions are arranged in a regular grid. For each grid cell, we use 1000 trials of our scenario to run both the 3D EKF and the 1D EIF on the same simulated stereo measurements. We count how often the 3D EKF has a lower absolute error than the 1D EIF after each fusion step. This overall percentage is the "win rate" for the 3D EKF.

For this scenario, we used parameters similar to KITTI with $B = 0.54\,\mathrm{m}$, $f = 700\,\mathrm{px}$, $c_x = 625\,\mathrm{px}$ and $c_y = 175\,\mathrm{px}$. Both filters use a stereo matching error standard deviation of $\sigma_d = 0.75\,\mathrm{px}$. For the pointing error, the 3D EKF uses $\sigma_u = \sigma_v = 0.5\,\mathrm{px}$, which means that we only have to inspect one quadrant due to the symmetry (or even 1D with either just X or Y offsets). Here, our 1D EIF does not use pose noise, while the 3D EKF uses $\sigma_{t_x} = \sigma_{t_y} = \sigma_{t_z} = 0.05\,\mathrm{m}$ for the translational pose error and $\sigma_{r_x} = \sigma_{r_y} = \sigma_{r_z} = 0.01\,\mathrm{rad}$ for the rotational pose error. Without pose noise, the 3D point would be too easy to triangulate for the 3D EKF.

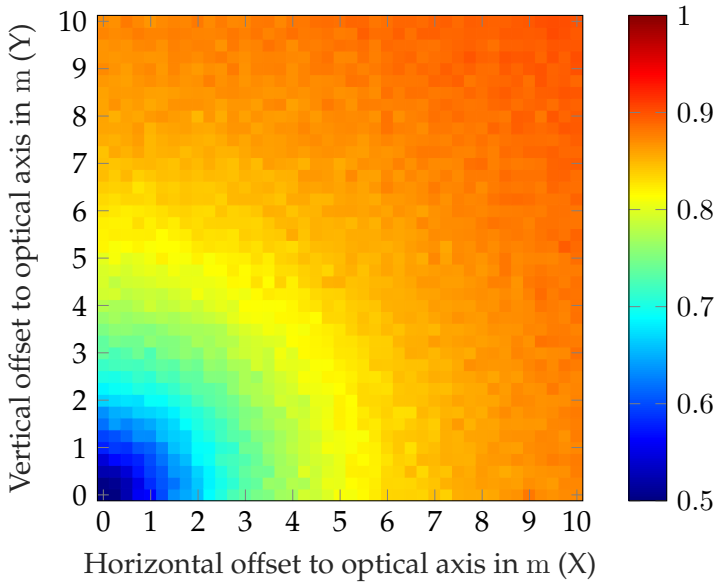


FIGURE 4.7: Win rate of the 3D EKF over the 1D EIF in an only forwards-moving stereo simulation. The X and Y axis denote the coordinate offset from the optical axis of the measured scene point.

We show the resulting grid map in Figure 4.7. Each cell is color-coded to show which filter performs better in comparison. The 3D EKF outperforms the 1D EIF consistently at almost every test scene point. Only scene points that lie on the optical axis (or are very close to it) are contested. The more we move away from the optical axis, the better the 3D EKF performance in comparison. To summarize, the 3D EKF can improve the fusion results compared to the 1D EIF even further by considering uncertainty along the U and V axis in image space.

4.6 Limitations

In this section, we will highlight the different consequences of our assumptions. We investigate error correlation, non-linear disparity sampling, static starting covariance and the Gaussian assumption for stereo matching errors.

4.6.1 Correlated Errors

One major assumption that we make in our fusion approaches is that disparity error estimates are uncorrelated over time. This assumption is valid when simulating disparity noise by sampling from a normal distribution (limited by the random number generation). In practice, however, this may not always be the case: If the vehicle stops, for example, the scene is static and we process frame after frame iteratively. Our fusion model would then assume that all points can be reconstructed with arbitrary depth accuracy, given enough time or frames. In reality, stereo matching algorithms would give very similar results each frame. For example, the disparity error of a scene point via a stereo matching algorithm of a particular pair of image patches may amount to $-0.5\,\mathrm{px}$ due to internal inaccuracies of the stereo algorithm for this constellation. It is unlikely that image noise alone can provide new patches in the following frames that will change this local error behavior significantly. This means that the stereo error would be correlated between frames to some degree for such a scenario.

Only image noise would make deterministic stereo results differ between frames and the reconstructed accuracy would converge to a certain limit. This means that we may be overestimating the accuracy of reconstructed points. In practice, using constant process noise for each frame, *i.e.* decoupled from the actual movement, will prevent this effect partly by lowering the convergence accuracy. Geyer *et al.* [Gey+06] describe the process of empirically estimating the correlation of the error between frames as "difficult, if not impossible". Still, detrimental effects of correlation are smaller while the vehicle is moving rather than being at a halt. We also hypothesize that correlated errors are less of a problem at shorter ranges since the image patches depicting a scene point can change more drastically between frames than from afar.

Correlated Gaussian Noise Modeled as an AR(1) process

Still, we can simulate correlated Gaussian noise and estimate what would happen if we had perfect knowledge of the correlation. For this, we use an autoregressive model of order 1 for the random process (AR(1) process). The motivation for this model is that the disparity error of a certain scene point can be quite similar from one frame to the next. For example, the corresponding image patches of the left and right image used in the stereo computation may be quite similar in frames 1 and 2. Frame 1 and frame 50, however, may have greater differences and the error is therefore less correlated. An AR(1) process approximates such a behavior.

An AR(1) process allows to compare how the error correlation affects the results of temporal fusion. Note that we can still get arbitrarily accurate over time with an AR(1) process as long as the correlation ρ is below 1. Thus, this model better describes a scenario with observations of a mobile camera over time rather than a perfectly still camera.

We refer to Box *et al*. [Box+15] for the derivation of the following equations. An AR(1) process with zero mean consists of a component that relates the previous state to the new state and a white noise process ϵ_t with zero mean and constant variance σ_ϵ. The process state X at time t is defined as:

$$X_t = \varphi X_{t-1} + \epsilon_t \tag{4.42}$$

We use a step width of $n = 1$ to model the use of the immediate next frame. $\varphi \in [0, 1]$ is a parameter of the model, which controls how much influence the immediate last state has on the current state. It remains constant here. The variance, covariance and the Pearson correlation coefficient can be computed as follows (see [Box+15] 3.2.3):

$$var(X_t) = \sigma^2 = \frac{\sigma_\epsilon^2}{1 - \varphi^2} \tag{4.43}$$

$$cov(X_t, X_{t+1}) = \frac{\sigma_\epsilon^2}{1 - \varphi^2}\varphi = \sigma^2\varphi \tag{4.44}$$

$$\rho_{X_t, X_{t+1}} = corr(X, X_{t+1}) = \frac{\sigma^2\varphi}{\sigma^2} = \varphi \tag{4.45}$$

In our case, the correlation between measurements $\rho_{X_t, X_{t+1}}$ is exactly the model parameter φ, which we can vary to simulate different correlations.

Application to Stereo Simulation

Next, we will apply the AR(1) process model to the simulation of stereo measurements with correlated errors. Here, a measurement z_t consists of the ground truth z_{gt} and additive noise from an AR(1) process:

$$z_t = z_{gt} + X_t \tag{4.46}$$

We assume that even the start value X_0 is part of an ongoing AR(1) process. This means we do not assume that X_0 would have a variance of σ_ϵ^2 because it is still influenced by unseen previous measurements. If we were to compute a histogram of all X_0 in the trials, we still expect the correlated variance of the AR(1) process at this point to be:

$$var(X_0) = \frac{\sigma_\epsilon^2}{1 - \varphi^2} \tag{4.47}$$

This correlated variance is bigger than σ_ϵ^2, which would be the variance in the uncorrelated case. In other words, our first measurement is already influenced by an ongoing process, which induces more variance. In Figure 4.8, we showcase the results of varying correlation factors for the raw measurements and our fusion. We are still able to lower the mean RMSE with our EIF over all trials, but are unable to achieve the same accuracy as in the uncorrelated fusion case (solid black line in Figure 4.8). Here, we are also overestimating the accuracy by assuming it to be the same as in the uncorrelated case.

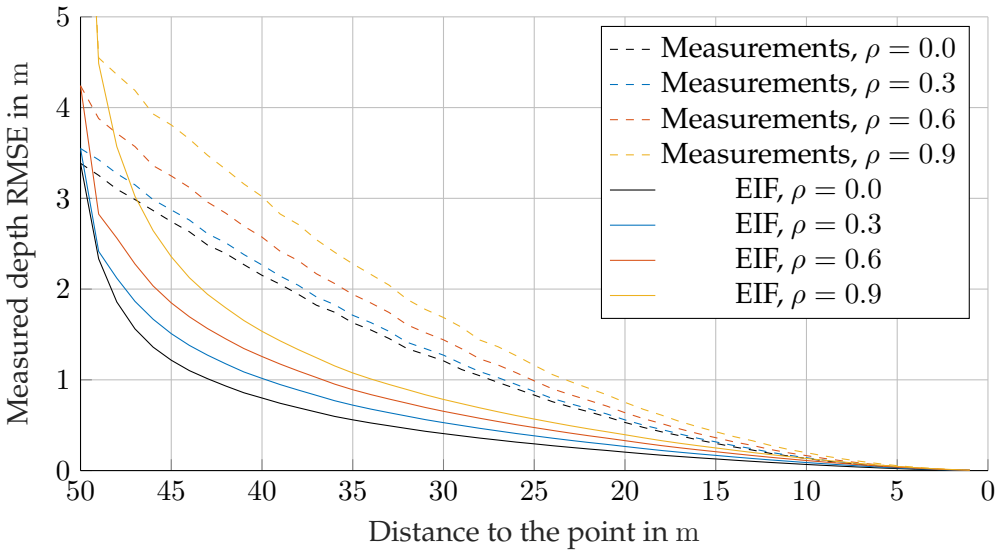
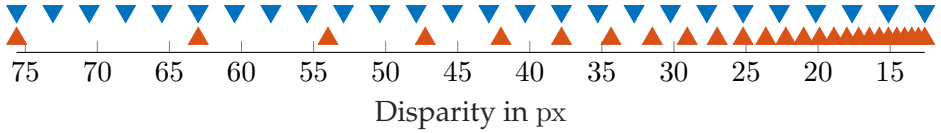


FIGURE 4.8: Average RMSE of 10 000 Monte Carlo trials of our EIF with correlated disparity error (frame to frame correlation $\rho = \varphi$ of the AR(1) process). One trial consists of one meter steps from 50 m to 1 m. Raw measurements are dashed and the corresponding fusion result has the same color.

4.6.2 Non-linear Sampling of Disparities

Consider a vehicle with a mounted camera that moves at a constant speed. This induces constant depth changes of static points, *e.g.* 1 m each frame in a car at a frame rate of 10 Hz and a velocity of 36 $\mathrm{km\,h^{-1}}$. Due to the inverse relationship of depth and disparity, a point seen from afar and then from up close will have a denser disparity sampling at first, which subsequently thins out as the vehicle gets closer. For example, a point from 50 m moving to 49 m with $B = 0.54\,\mathrm{m}$ and $f_x = 700\,\mathrm{px}$ will change by 0.15 px in disparity space, but from 10 m to 9 m it will change by 4.2 px.

To demonstrate the effects of disparity sampling further, we measure the same scene point with a simulated stereo camera with $B = 0.54\,\mathrm{m}$, $f_x = 700\,\mathrm{px}$ and $e_d = 0.5\,\mathrm{px}$ as we did before. We simulate camera movement along the optical axis from 30 m to 5 m in depth (which corresponds to 75.6 px to 12.6 px in disparity) with 26 steps (including start and end point). Along this interval, we use two sampling variants. The first variant distributes the disparity measurements equally along the



(a) Disparity of sampled points. Right-most point is the starting point.

(b) Depth of sampled points. Left-most point is the starting point.

FIGURE 4.9: Sampled points of both sampling variants. **Blue**: Constant disparity changes. **Orange**: Constant depth changes.

interval, while the second variant does the same for the depth measurements. The first scenario has a constant *disparity* change of $2.52\,\mathrm{px}$ and the second scenario has a constant *depth* change of $-1\,\mathrm{m}$ per step. In Figure 4.9, we compare both sampling variants in terms of disparities and depths. The blue markers denote the first scenario and the orange ones the second scenario. The constant depth change scenario resembles a vehicle (with a front-facing stereo-camera) in the common scenario of moving at a constant speed, while the constant disparity change scenario resembles a smooth deacceleration process of a vehicle.

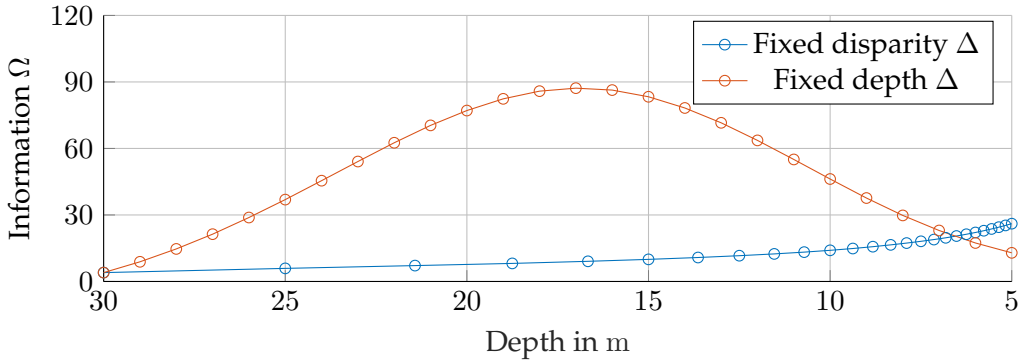
In Figure 4.10, we show the results of our 1D EIF on both sampling variants in terms of the information, reweighting factor, disparity error and propagated depth error. The reweighting factor is the weight of the propagated information during the fusion with a new measurement. The disparity error is derived from the information and the depth error is computed according to the stereo matching error model. For the constant depth change variant, the information of a point during this movement will form an inverted U-shape (see Figure 4.10a), so that the corresponding estimated standard deviation will form a U-shape (see Figure 4.10c). One could consider a confidence-based thresholding of the filtered point based on the information (or variance): For example, we may not want to display inconfident points because of potential outliers and only display points with an estimated disparity error e_d lower than some threshold. This thresholding, however, turns out to be problematic here since close stereo points, which provide excellent depth estimates, may be omitted from the reconstruction since their estimated information is too low (or their estimated standard deviation too high). For example, if we would only accept points with $e_d < 0.2\,\mathrm{px}$ in our reconstruction, then we can see in Figure 4.10c that we would accept a scene point at $26\,\mathrm{m}$, but we would reject it at $7\,\mathrm{m}$. Thus, near points may be omitted. The constant disparity change sampling variant does not have this problem. Still, we can see in Figure 4.9b that we perform better than unfused stereo without thresholding, even with constant depth sampling. Note that thresholding the

estimated depth error instead will simply result in far depth estimates to be omitted, which proves to be undesirable for visualization purposes.

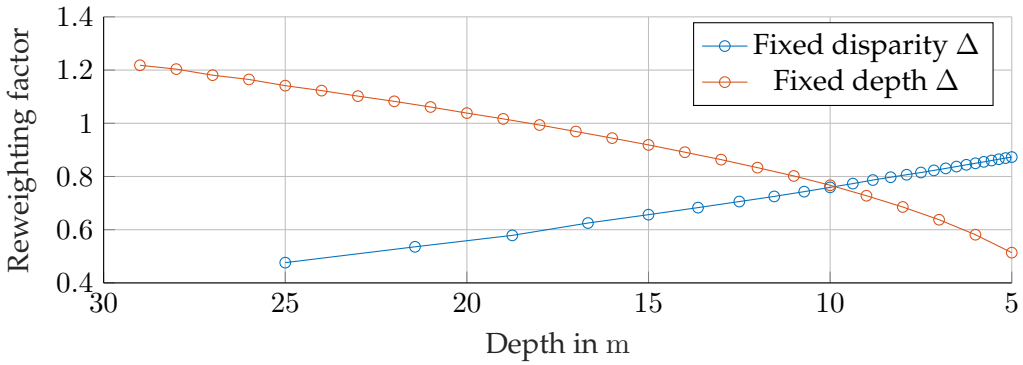
The reason for the inverted U-shape of the information is the following: At first, the fusion will build up information (see Figure 4.10a) because the reweighting factor for the fused history is greater than 1 until a depth of $19\,\mathrm{m}$ as shown in Figure 4.10b, where it will begin to accelerate downwards with a factor smaller than 1. This means that we are putting less and less weight onto the fused history of the point and put even more weight onto the new and near measurements during fusion. This happens due to the increasing changes in disparity in that case (see Equation 4.8) and makes sense, because a scene point that is much closer can be measured more accurately than when it was far away. Therefore, the information will almost return to the baseline, *i.e.* the starting information. For the constant disparity change variant, we can see that the information and reweighting factor is increasing monotonically, but the reweighting factor starts at a very low value comparatively.

Essentially, a scene point observed from afar needs much more information to be on par with fresh observations when propagated to a closer depth due to the stereo error. This means that multiple fused observations of a far depth will quickly lose their gained relative accuracy when propagated to be closer to the camera. Note that the information in the fixed disparity change scenario accumulates, while it will accumulate and then decay in the fixed depth change scenario.

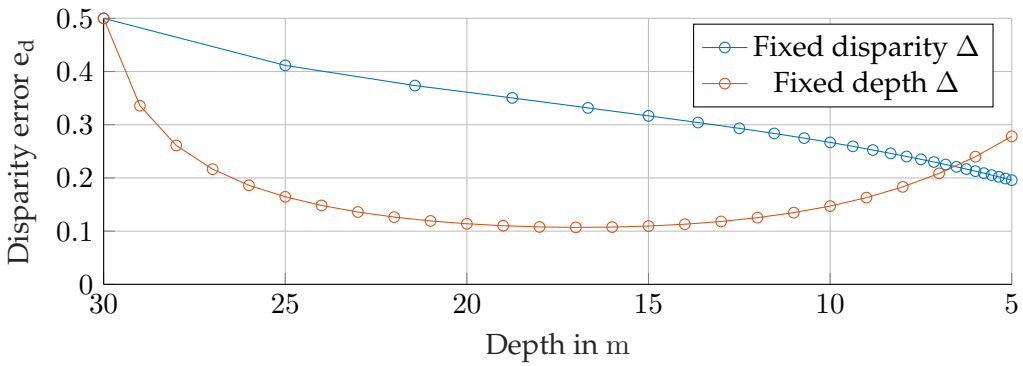
The 3D EKF is similarly affected by this, but since the covariance also incorporates the u and v dimension, the effect is smaller (for scene points not lying on the optical axis).



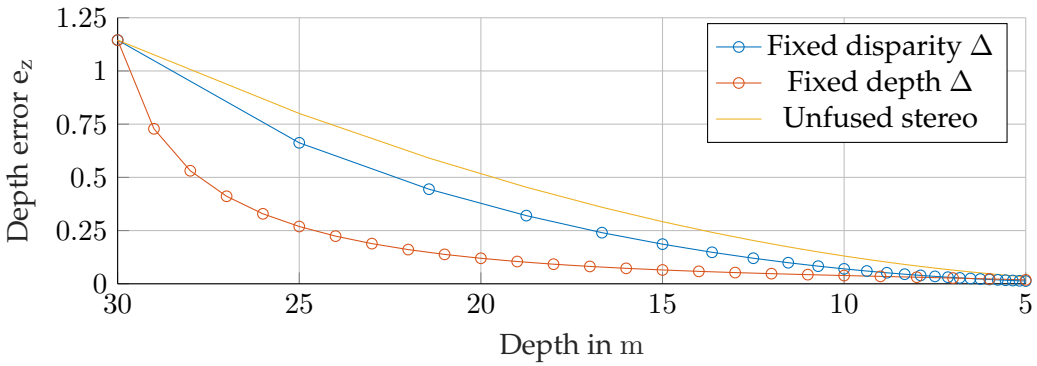
(a) Information over depth.



(b) Reweighting factor for the information propagation over depth.



(c) Disparity error depicted over corresponding depth.



(d) Propagated depth error from disparity error.

FIGURE 4.10: Information, reweighting factor, disparity error and depth error for both sampling variants. Note that the information Ω can be directly transformed to the disparity error e_d via $\Omega = \sqrt{\frac{1}{e_d}}$. Also note that e_z is computed from e_d via Equation 4.1.

4.6.3 Constant Starting Covariance

Our approach is independent of any particular stereo matching algorithm. Nevertheless, none of the used algorithms provide explicit confidence information like a variance measure. Even though there are some general stereo confidence measures (see below), we simply assumed a constant starting covariance for each pixel ($\mathbf{Q}$) in our 3D EKF. We could change $\mathbf{Q}$ to accommodate for different σ_d^2 (and σ_u^2/σ_v^2) as input. For example, pixels at homogeneously textured areas or horizontal depth edges should have a lower confidence than pixels at vertical depth edges in the typical horizontal stereo setup. In essence, we may overestimate the variance of some pixels and also underestimate the variance of other pixels. Accurate confidence information could potentially improve the results of 3D reconstruction, provided that there is an accurate mapping from the confidence measure to σ_d^2.

Stereo confidence measures have been vastly researched and are widely used. Their intended use is to improve, filter or fuse existing disparity maps. For that, many measures have been proposed (like using the matching costs) and evaluated on benchmarks, considering their ability to predict the error or stereo matching accuracy (see [EMW02; HM12]). One result from these studies is that some measures are better than others, but not one single measure outperforms all in every scenario. A combination of multiple measures (an ensemble), outperforms a single confidence measure (see [HK12; HNK13]). Using a specific confidence measure as an add-on shows improvements in terms of robustness or quality in existing (conventional) stereo algorithms (see [PY15; PGS13; Ma+18]). Recently, deep learning confidence measure approaches outperform the aforementioned ensemble measure (see [Tos+18]) or can enhance traditional stereo matching (see [SP16]). In the future, a deep learning stereo matching may also provide a confidence map along with the disparity map.

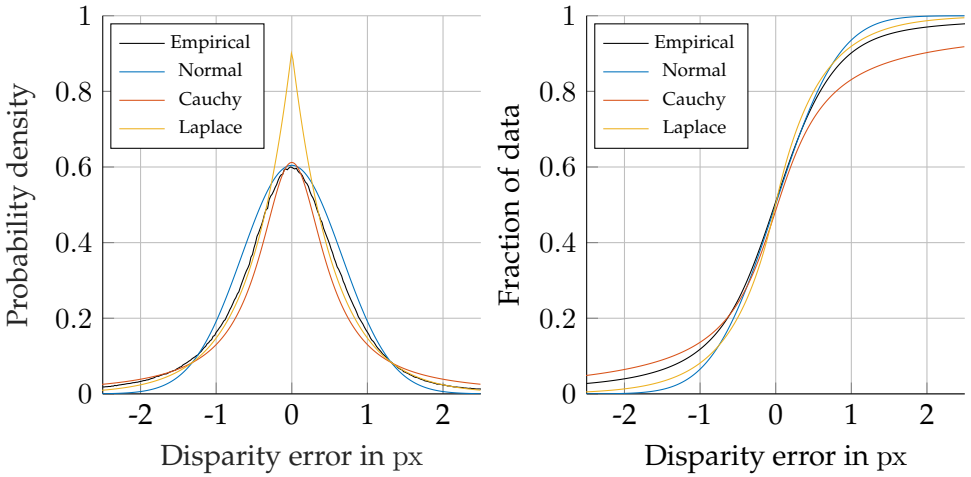
4.6.4 Stereo Matching Error as Gaussian Noise Assumption

For Kalman filters used in the stereo vision context, one key assumption is Gaussian stereo matching noise, *i.e.* Gaussian disparity (or depth) errors. As previously mentioned, the real stereo matching error in disparity may not be normally distributed. This inaccuracy would mean our fused (co-)variance estimate may be biased and the resulting disparity estimate skewed due to non-adequate weights in the fusion. If the Gaussian assumption is not given, *e.g.* if the distribution is actually skewed or multimodal, our fusion algorithms might not improve the accuracy of the measurement because of an inherent bias.

Popular stereo matching benchmarks such as the "Middlebury Stereo Evaluation" [SS02] or "Stereo Evaluation 2015" from the KITTI Vision Benchmark Suite [MG15] do not report the error distribution directly. Sibley *et al.* [SMS07] use the assumption that the disparity error is normally distributed and provide evidence using corner features and a synthetic checkerboard scene. Similarly, Matthies and Grandjean [MG94] found the disparity distribution to be Gaussian-like when computing the disparity

by minimizing the sum of squared difference in a search window. Sebe and Lew [SL00] found that a better fit for their case is a Cauchy function when using the sum of absolute differences or sum of squared differences with real world ground truth data. Wedel and Cremers [WC11] propose to approximate the disparity distribution with a Laplace distribution.

In Figure 4.11, we show the empirical disparity error distribution of ELAS using the supplied "ROBOTICS" parameter set on the training set of the KITTI 2015 stereo dataset. Both empirical probability density functions and cumulative distribution functions are shown. We adjust the empirical probability density function to have zero mean (*i.e.* removing some bias of the error). We also plot different plausible distributions with manually chosen parameters: A normal distribution (with $\mu = 0\,\text{px}$ and $\sigma = 0.66\,\text{px}$), a Cauchy distribution (with $x_0 = 0\,\text{px}$ and $\gamma = 0.52\,\text{px}$) and a Laplace distribution (with $\mu = 0\,\text{px}$ and $b = 0.55\,\text{px}$).



(a) PDFs of the disparity error.

(b) CDFs of the disparity error.

FIGURE 4.11: Probability density functions (PDFs) and cumulative distribution functions (CDFs) of an empirical error distribution and manually fit distributions on the KITTI 2015 stereo dataset.

Computing an empirical standard deviation for ELAS results in $\sigma = 1.31\,\text{px}$, which contradicts the standard deviation of the manually fit normal distribution. The Cauchy distribution is heavy-tailed and has no variance. The empirical error distribution also has long tails, which influence the computation of the error variance due to the squared error distance heavily. The Cauchy distribution might be the best single fit. In Figure 4.11b, it is clearly visible that the normal distribution reaches zero and one very fast at the tails, which makes it improbable that the true error is normally distributed. Additionally, the true distribution may be a combination of distributions, *e.g.* a normal distribution for the pure disparity error and a uniform distribution because of disparity outliers, which cannot be modeled by a single distribution model.

Still, as long as this distribution is unimodal, symmetrical and unbiased, we can eventually reach a good estimate with a fusion model based on normal distributions,

but not the optimal estimate. Nevertheless, this noise attribute is highly dependent on the stereo matching algorithm itself and the transformation to the ground truth coordinate system in practice, *i.e.* the calibration between the depth sensors.

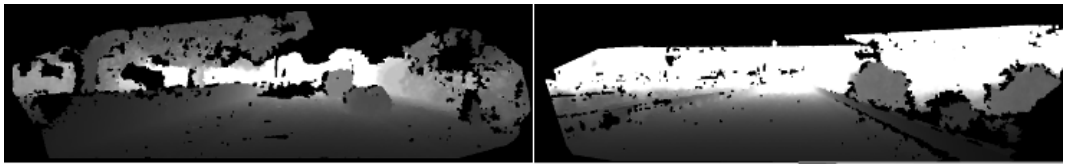
If the disparity error distribution is biased ($\mu \neq 0$) or skewed, then there exists an upper bound on the fusion accuracy. It is only possible to increase the precision in this situation. This in effect means that we can stabilize the reconstruction of a scene point with fusion, but some disparity dependent error always remains. For visualization purposes, a consistent reconstruction may still be achievable, even though the disparity estimates may be slightly biased towards positive or negative errors.

4.6.5 Inaccurate Calibration between Stereo Camera and Ground Truth

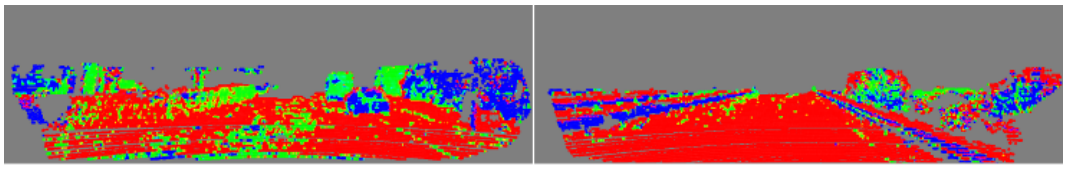
Even when assuming that a certain stereo matching algorithm exhibits Gaussian noise for the disparity error, an inexact transformation to the ground truth depth sensor measurements would yield a systematical error overestimation in a quantitative error evaluation. Thus, refining an existing calibration in practice could improve quantitative results without changing the actual stereo depth estimation process. To prepare for the quantitative error evaluation, we recalibrate the KITTI 2012 VO dataset. We take the transformation from the Lidar system to the stereo camera and optimize it by finding an offset which improves a "bad pixel" error metric over a KITTI sequence. The method is described in detail in our technical report [HNS19a] and consists of maximizing the area under the curve near zero with a genetic algorithm. Here, we only briefly present the results of our improved calibration.

We show two frames of different scenes of the KITTI VO dataset in Figure 4.12. We compare the disparity of a single full-frame ELAS estimate to the Lidar ground truth with the original calibration and our new calibration. The original calibration exhibits a clear bias on the road surface and horizontal structures, while vertical structures like the hedge and vehicles show less bias. The recalibration greatly diminishes the road surface bias and keeps other structures intact. Comparing the error distributions of all KITTI VO sequences in Figure 4.13, we can see that the bias is eliminated and the distribution is now more symmetric and unimodal for each sequence. The resulting calibration works for other stereo matching algorithms too, *i.e.* recalibrating with ELAS also improves other approaches, which means that we are not overfitting. The exact parameters are disclosed in the technical report [HNS19a].

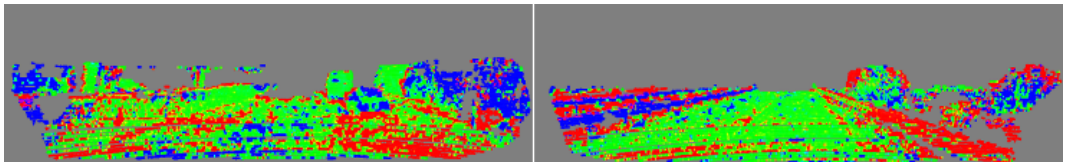
(a) Left color camera RGB image.



(b) Left color camera ELAS depth image.

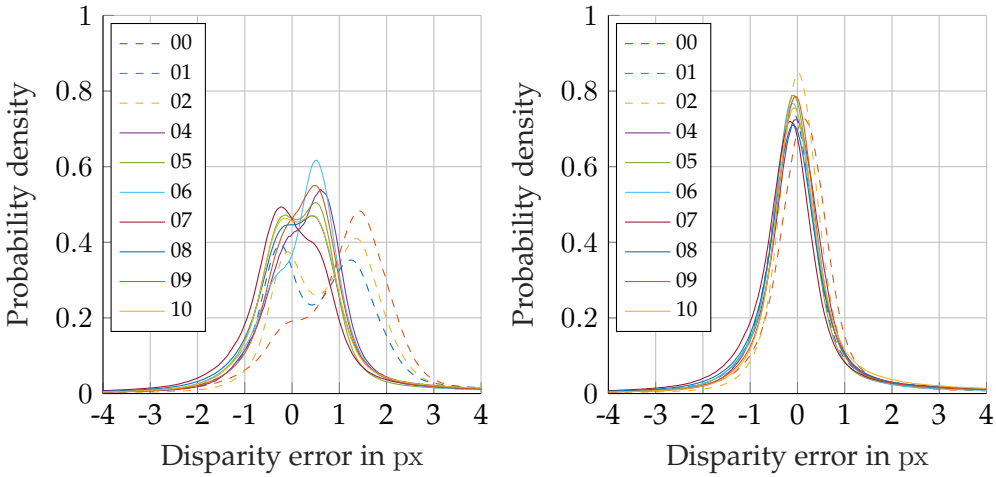


(c) Disparity comparison of stereo to Lidar measurements with original calibration.



(d) Disparity comparison after applying our new calibration.

FIGURE 4.12: Comparison of disparities between ELAS and the ground-truth on frame 08-430 (left) and frame 01-1100 (right) of KITTI VO. Red corresponds to positive disparity errors and blue to negative ones. Green represents good matches below an absolute pixel error of 0.5 px.



(a) ELAS with original calibration.

(b) ELAS with our recalibration.

FIGURE 4.13: Disparity error distributions of ELAS over the KITTI visual odometry training sequences without and with our recalibration.

Chapter 5

3D Reconstruction with a Front-Mounted Stereo Camera on a Vehicle

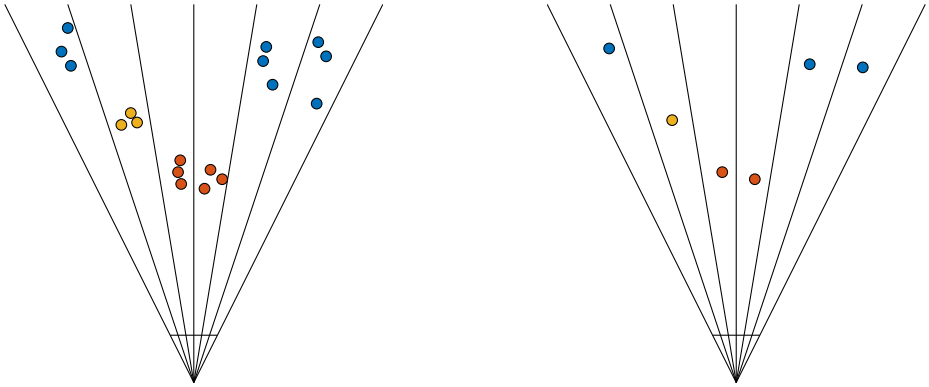
5.1 Introduction

The previous chapter dealt with *how* we fuse scene point estimates. In practice, it is obviously necessary to consider *what* to fuse as well. Here, we demonstrate how we deal with the *what* and detail our 3D reconstruction approaches. For that, we only consider the fusion of scene points over multiple frames. This means that we are concerned with temporal fusion of observations from a single stereo camera rather than using measurements from different cameras like Gallup [Gal11], for instance.

We chose a single stereo camera for this task for several reasons. The main reason is that the stereo camera is already built-in in modern cars for driver assistance purposes, which makes this camera readily available. Furthermore, a single stereo camera is cheaper, easier to calibrate, synchronize and integrate in an automotive system than multiple cameras. Naturally, the major drawback of a single stereo camera is the limited field of view. A temporal 3D reconstruction over multiple frames with a single stereo camera while driving will contain scene points that have moved outside the FoV of the camera. The points that left the camera frustum are essentially frozen because it is likely that they will not change anymore until you pass the scene again with the vehicle, but they can still be rendered to achieve a reconstruction outside the camera frustum. We can therefore greatly extend the FoV of the camera by considering previous frames with static 3D scenes.

Temporal fusion of stereo estimates requires pixel warping, which in the 3D reconstruction context means to transform the pixels of one image to another frame. Each pixel in the previous frame has an associated disparity/depth and position in image space. Using this knowledge, we can undo the image projection via the known position, orientation and calibration matrix of the camera. This is called the backprojection step, in which we transform image space coordinates to 3D world coordinates. Next, we use the updated camera pose of the new frame and the

calibration matrix to project the 3D point to a 2D image point (with an associated depth) in the new frame.



(a) Raw measurements. (b) Per-pixel fused measurements.

FIGURE 5.1: Pixel fusion along the volumetric ray of each pixel.

A main motivation for our approach is that salient objects like traffic signs appear multiple times in the reconstruction when naively combining all measured 3D points to a dense point cloud over multiple frames. Due to the stereo error, each observed duplicate will have some variation. For instance, Figure 1.4 shows a sample scene of KITTI VO with ELAS as seen from a perspective different than the original optical axis over multiple frames. Every stereo measurement throughout the sequence is rendered (with maximum point size in OpenGL). We assume that the depth error is the main source of the positioning error of a 3D point, which is apparent in the example. Our goal is to enforce only a single reconstruction at the same approximate depth levels ("layer"), which we propose to accomplish by merging points along the viewing ray volume of each pixel. As a result, we can attain to observe only one single traffic sign at a single layer, while also improving depth accuracy via temporal fusion. We illustrate this concept in Figure 5.1. Observations from three frames (warped into the current frame) are fused in camera space along the ray volume of each pixel. The color of the dots represents separate scene objects.

The warping step alone may cause false correspondences during fusion, but we punish rotational and translational camera pose errors in our filters. Alternatively, Alcantarilla *et al*. [ABD13] compare patches around the pixel of frames to dismiss pixels with insufficient photometric consistency. Warping pixels corresponds to an implicit dense scene flow computation of a static scene: With warping, we have to estimate the 3D displacement for every scene point in the previous frame to the next frame due to camera movement, including points that are now outside the frustum.

Note that we use forward mapping to warp the pixels from the previous frame to the new frame. Alternatively, one could warp the new frame to the previous frame to fuse pixel estimates, which is called backwards mapping. Backwards mapping is usually used in virtual view synthesis on a single image to generate a virtual camera image without pixel sampling holes (see Scharstein [Sch99]). Still, conventional stereo matching algorithms may not deliver a $100\,\%$ complete depth map and it is impossible

to backwards map empty pixels to the previous frame. This prevents us from filling potential holes with the previous frame. Furthermore, our later introduced tube meshing approach relies on forward mapping to check whether each pixel lies in the frustum of the new camera pose. Even though backwards mapping might be worthwhile in other contexts or approaches, we do not investigate the effects of using it any further.

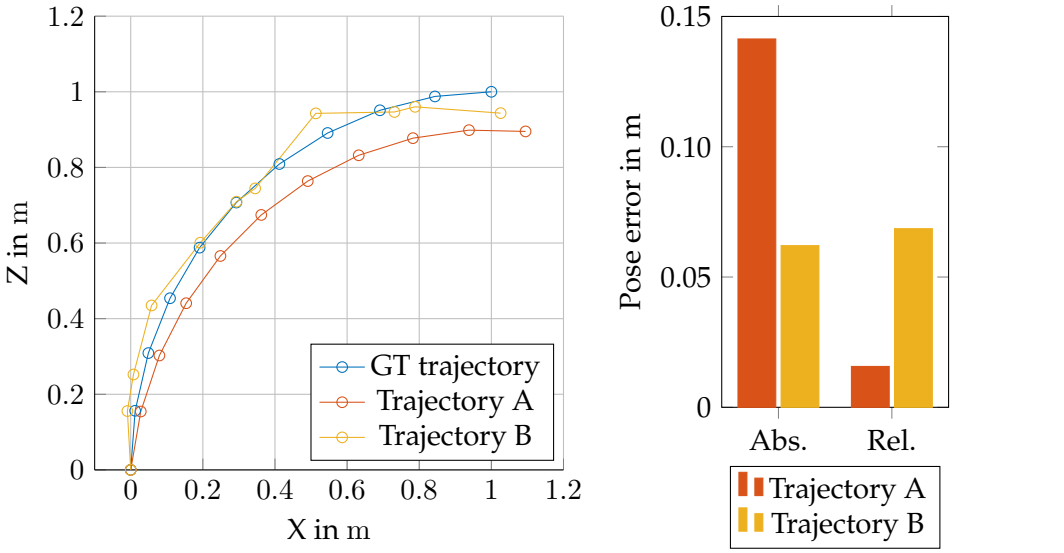
Due to the camera movement in the scene, warping pixels to a new frame alone results in multiple estimates per target pixel. For example, a tree in the previous image may move partially in front of a house facade in the next image, which was not covered in the frame before. This area now has multiple pixel estimates, *i.e.* we need to be able to handle more than one depth layer in the image. Additionally, we have to decide which layer from the warped previous estimate corresponds to the current estimate because we want to prevent false temporal correspondences.

In the following sections, we develop a point-based reconstruction framework, which tackles the aforementioned problems with a 1D extended information filter. Then, we extend the point-based reconstruction framework to a mesh-based reconstruction using a 3D extended Kalman filter. In the end, we propose a method to replace sky pixels in the reconstruction with an immersive virtual sky, which can serve as a background to our reconstruction.

5.2 Point-based Reconstruction

As input to our reconstruction we need both calibrated stereo image pairs and the respective left camera pose. In practice, we may use the already provided highly accurate pose obtained from sensor fusion that is used for vehicle localization (and AR use cases). This means we may not have to rely solely on VO if the camera is calibrated relative to the vehicle position. Still, we acknowledge that VO does not need a synchronization mechanism and that the reconstruction itself can potentially be used to refine the pose estimates. Furthermore, absolute pose drift may not matter that much for our immersive use case since we can dismiss part of the reconstruction as the camera passes by the geometry. However, the relative pose error should be low and consistent, *i.e.* it should not contain jumps, so that the depth maps between frames align in 3D.

In Figure 5.2, two different example camera pose trajectories are shown and compared against the ground truth in terms of their translational error. We compute the absolute error as the distance between the last poses and the relative error as the mean distance between the true pose change and the estimated pose change. Trajectory A is smooth, while trajectory B moves more erratic and follows the ground truth more closely. The absolute pose error of trajectory B is less than that of A. But the relative error of trajectory A is much lower than that of trajectory B, which is a desired property of the provided pose estimates. This example shows that it might be worthwhile to accept some absolute error for a smoother camera trajectory.



(a) Camera pose trajectories.

(b) Absolute error and mean relative error.

FIGURE 5.2: Comparison of two trajectories against a ground truth trajectory.

Pose jumps in the reconstruction would cause user irritation and possibly nausea when viewed live on an HMD. Also, the reconstruction would not align with pose jumps since the stereo measurements are quite consistent among frames, which would cause visual duplication artifacts. Consequently, using pose estimates of a SLAM system would produce jumps at loop closures. It would mean an expensive step to reconstruct the sequence anew with updated poses, which makes SLAM systems not an adequate choice for our application.

5.2.1 Overview of Point-based Pipeline

Figure 5.3 shows the point-based reconstruction pipeline of our system. For each frame, the whole pipeline is run through. As input (in red), we have the estimated current camera pose and the current stereo matching result (disparity map). We output 3D world points after one iteration, which we visualize as either in the active or passive point cloud (in green). The 3D world points are empty at the start. A 3D point consists of its 3D position, RGB pixel color and additional data needed for the filter. For instance, the 1D EIF needs the previous disparity and the previous information as input.

5.2.2 Warping

We divide the reconstruction into two regions as shown in Figure 5.3: Active and passive. The active region contains points that are still inside the camera frustum. Only active points need to be considered when integrating the new measurement from the current frame into the reconstruction during an iteration. The passive region contains points that are outside the frustum. We assume that these points are not

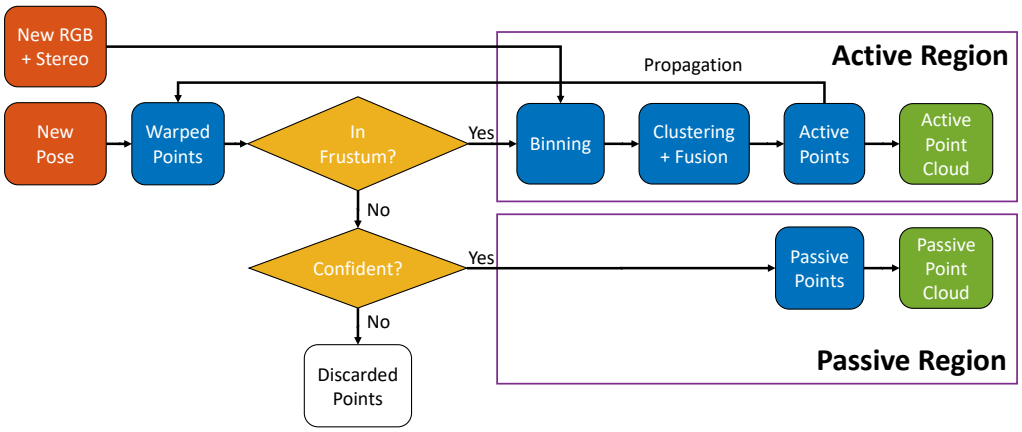


FIGURE 5.3: High-level overview of our point-based reconstruction pipeline. Red represents input, green rendered output and white removal.

seen anymore and are therefore frozen. This assumption is generally true when driving forwards, but not necessarily true when the pose trajectory is very snakelike. The passive region is still rendered because it enlarges the reconstructed scene. For instance, this allows us to render the road directly below the input stereo camera, even though the camera cannot see it anymore.

The reconstructed world points observed in the very first iteration are forming the active region at the start. With the new pose in the next iteration, we can project the active points to the current image frame ("Warped Points"). If the point lands outside the image, *i.e.* outside the camera frustum, we assign it to the passive region. However, we discard inconfident points. The confidence check is satisfied according to a disparity error threshold T_d for the estimated information. Both checks are depicted in yellow in Figure 5.3.

For points that are inside the new camera frustum, we use the respective propagation equations to update their information. For that, we have to rely on the previously saved information and disparity in the point.

5.2.3 Binning

After the warping, we bin the projection of the remaining active 3D points in image space to pixel bins ("Binning"). A bin is an unsorted list of all points projected onto a certain pixel with potentially different depths. We bin points to integrate the newly measured disparity map and to handle pixel collisions due to multi-layered depths in the scene by *e.g.* occlusions due to movement. We propose three different ways to structure the binning, which we will discuss in the following.

Nearest Neighbor

The first approach is adding the point to the nearest pixel bin, which we call "nearest neighbor" binning. One problem is that Moiré patterns are introduced when the

camera moves forwards, *i.e.* moving towards the optical axis. This is always the case for a front-facing stereo camera on a vehicle except for certain situations like parking maneuvers. Figure 5.4 shows the result of warping a depth map to the next frame using nearest neighbor binning. Those patterns exist because the points have no volume or area, but have to cover a larger image space surface than before due to the forwards movement. The 3D point cloud essentially thins out. Notice that the small pillars on the right are now fragmented. Also, when a pixel is located on the same horizontal level as the bin centers A and B and just a tiny amount closer to B, then only bin B receives the pixel. This is unfair since bin A almost has the same distance to the pixel as bin B.

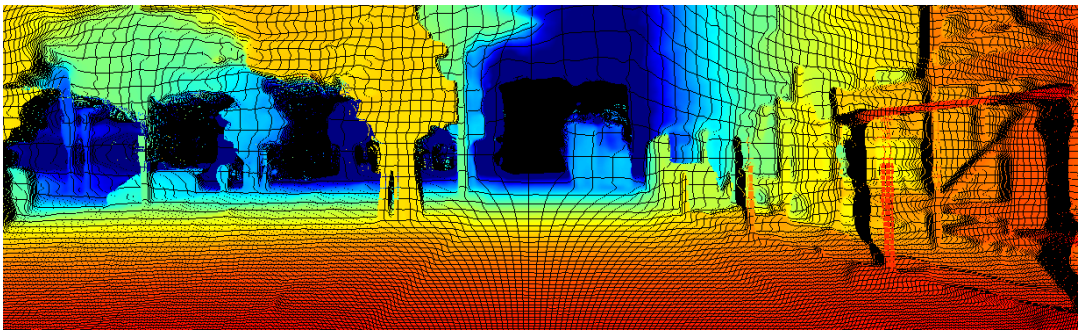


FIGURE 5.4: Resulting colorized depth image of warping a frame to the next one on KITTI VO 06 using GA-Net.

Inverse Distance Weighting

An approach that prevents distributing the pixels according to a winner-takes-all scheme has to take into account the subpixel distance from the projected point to each surrounding pixel center. For this, we splat a projected point onto the four nearest pixel centers by distributing the information according to computed weights in this 2×2 window. We split the original information of the point according to its weights, which means that we do not artificially inflate the information.

Each weight w_i ($i = 1, \ldots, 4$ denotes the surrounding pixel) is computed via inverse distance weighting with an exponent of 2, which corresponds to the Euclidean pixel distance $d^2(p_0, p_i)$ from the local subpixel coordinate p_0 to each of the four corner pixels p_i:

$$w_i = \frac{\frac{1}{d^2(p_0,p_i)}}{\sum_{j=1}^{4} \frac{1}{d^2(p_0,p_j)}} \tag{5.1}$$

Figure 5.5 showcases an example of inverse distance weighting. The red circle corresponds to a projected point at the relative position $p_0 = (0.25,\ 0.75)$ that distributes its information to the surrounding pixel centers according to the computed relative weights.

The position in image space offers us two variants: We could use the pixel center of the bin as the new image position or use the accurate subpixel position of that

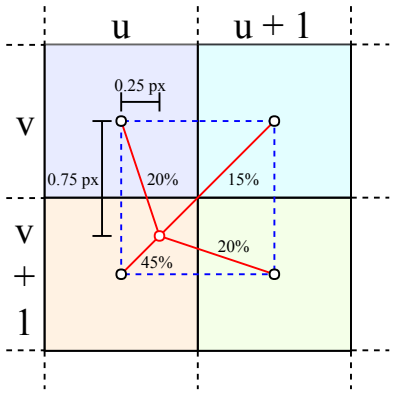


FIGURE 5.5: Example of weighting with inverse subpixel distances.

point, which means that an element inside a bin can have a UV coordinate that is not contained in that particular bin. We call this variant "sticky" inverse distance weighting. In Figure 5.5, only the bottom left pixel bin actually contains that point in its pixel area. Still, the weight would be very small for the other bins, which would shift incoming new measurements (located at the pixel center of that bin) during fusion slightly towards a subpixel position outside the bin.

Pixel Size Weighting

In the previous two binning approaches, the projected point in image space has no explicitly modeled area. Given a pixel with an edge length of $1\,\mathrm{px}$ and an area of $1\,\mathrm{px}^2$, if we reproject this pixel into a new frame, the physical dimensions will change due to camera movement. The idea of this approach is to distribute the information of a point to the bins according to the covered pixel area in image space.

The gain or loss in (horizontal) pixel length can be derived by backprojecting the pixel edge positions X_{left} and X_{right} to 3D in camera space and then projecting them to the current camera frame (essentially warping the pixel edges) where we can measure the distance between those points. First, we compute the 3D position of the pixel edges:

$$X_{left} = \frac{(u - 0.5\,\mathrm{px} - c_x)}{f_x} \cdot z_{old} \qquad X_{right} = \frac{(u + 0.5\,\mathrm{px} - c_x)}{f_x} \cdot z_{old} \tag{5.2}$$

u corresponds to the horizontal image coordinate of the point, c_x is the horizontal image center and f_x the horizontal focal length. z_{old} denotes the depth value in the previous camera frame. Then we compute then horizontal positions of the edges u_{left} and u_{right} in the new frame via projection:

$$u_{left} = \frac{X_{left}}{z_{new}} f_x + c_x = \frac{\frac{(u - 0.5\,\mathrm{px} - c_x)}{f_x} \cdot z_{old}}{z_{new}} f_x + c_x = (u - 0.5\,\mathrm{px} - c_x)\frac{z_{old}}{z_{new}} + c_x \tag{5.3}$$

$$\Rightarrow u_{right} = (u + 0.5\,\mathrm{px} - c_x)\frac{z_{old}}{z_{new}} + c_x \tag{5.4}$$

Next, we compute the difference between the new edges to get our warped pixel length:

$$
\begin{aligned}
u_{length} &= u_{right} - u_{left} && (5.5)\\
&= \left((u + 0.5\,\mathrm{px} - c_x)\frac{z_{old}}{z_{new}} + c_x\right) - \left((u - 0.5\,\mathrm{px} - c_x)\frac{z_{old}}{z_{new}} + c_x\right) && (5.6)\\
&= 0.5\,\mathrm{px}\frac{z_{old}}{z_{new}} + 0.5\,\mathrm{px}\frac{z_{old}}{z_{new}} && (5.7)\\
&= \frac{z_{old}}{z_{new}} && (5.8)
\end{aligned}
$$

u_{length} denotes the horizontal pixel length of the old pixel warped to a new depth. The vertical pixel length v_{length} can be derived analogously if the pixels are not square, *i.e.* the pixel width and pixel height differ.

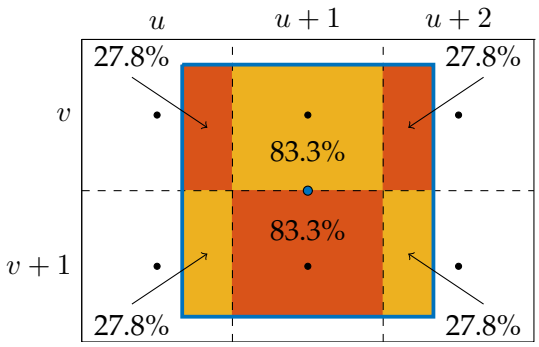


FIGURE 5.6: Individual percentage of the covered area of a warped pixel (blue outline) on pixels of the new image.

The weight in this approach corresponds to the covered area of the splatted point in the new frame. For example, if a current frame's pixel is covered completely by a pixel splat, then it will receive $100\,\%$ of the information. In Figure 5.6, we show a sample situation, in which the pixel center lands on the edge of two pixels and the pixel area covers six pixels. All the six hit pixel bins will append this point with respective weights to their list. To compute the weights, we first compute the new pixel edge length and check for collisions of the projected rectangle with the pixel bins in the new frame. For that, we compute the horizontal and vertical length and check how much of the pixel is covered by the splat. Then, the point is added to the bin with an applied discount to the information according to the coverage ratio.

Integrating New Disparity Measurements

After weighting, we integrate the newest stereo matching disparity estimate to the corresponding pixel bins with starting information $\frac{1}{e_d^2}$. The stereo matching method itself is freely exchangeable. Note that none of the used stereo matching algorithms output confidence measures, which we could however easily integrate as the starting information.

5.2.4 Clustering

As the vehicle is moving, points with different corresponding depth layers will be contained inside a bin and should not be merged, *e.g.* a traffic sign moving in front of a house in the reconstruction as seen from a new frame. For that, we employ a greedy unsupervised 1D clustering method to make sure that we do not fuse the points from different depth layers ("Clustering" in Figure 5.3). Otherwise, this will result in obvious reconstruction errors. Our clustering approach consists of iterating through all the points in the bin and checking, via a Z-test, if their disparities could originate from the same population, *i.e.* the same depth layer. If they do, we fuse both elements. If not, we handle both bin elements as separate clusters in a list and continue in the iteration until no unprocessed points remain inside the bin.

Two points "fit" together if they satisfy a Z-test using the disparities as expected values μ_1 and μ_2 with variances σ_1^2 and σ_2^2 and the population sizes n_1 and n_2:

$$Z = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \quad \text{with} \quad n_i = \frac{\Omega_i}{\Omega_{start}} = \frac{\sigma_{start}^2}{\sigma_i^2} \tag{5.9}$$

The population size n_i is implicitly stored in the information, which can be recovered by dividing the information through the original starting information. The Z-test succeeds if the Z-score is below a clustering threshold $T_c = 3.0$. This corresponds to a $3{\cdot}\sigma$ confidence interval, which is commonly used and delivered good empirical separation results on synthetic data. Figure 5.7 shows one example of a potential fusion situation in a pixel bin with three different depth layers.

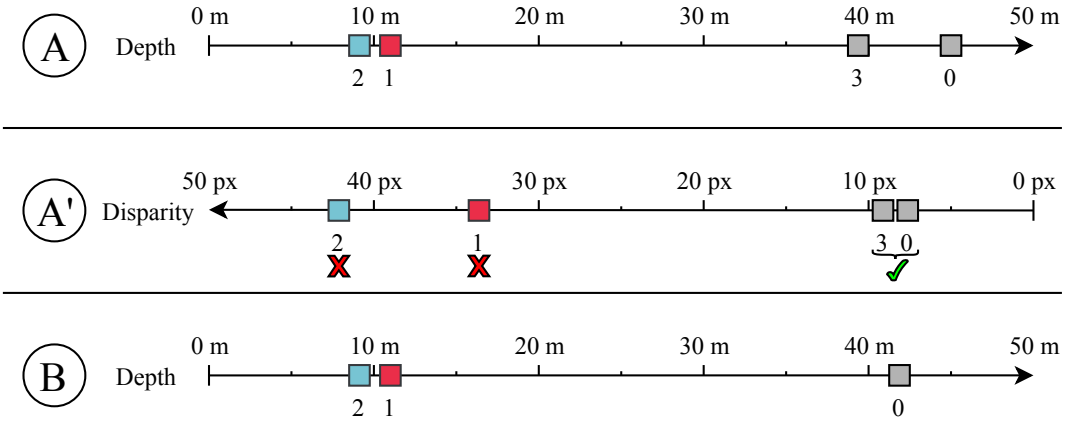


FIGURE 5.7: **A**: Depths of points with associated indices in a bin before the fusion. **A'**: **A** in disparity space. Note the inverted scale. Point 0 and 3 satisfy the Z-test and are fused. **B**: Resulting fused bin content.

The algorithm is greedy and may not result in the optimal solution in the first iteration. Sometimes, it will keep three points of the same depth layer separated in two clusters because of the fusion order, which is possible but not common since the measurements need to have some distance to the true mean. This might not matter over multiple frames since other measurements closer to the true mean can merge the

two points back to one layer. Or outlier depth layers will either lose information after a while without fused observations or can be dismissed according to our freespace check, which we will discuss later.

5.2.5 Fusion

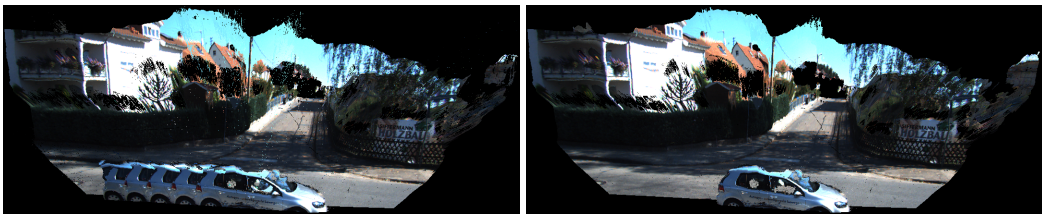
We compute the fusion of two pixels as shown in Equation 4.10 ("Fusion" in Figure 5.3). For the color, we adopt the pixel color of the newest point, which preserves color intensity. This also enables us to directly reflect color changes of static objects like traffic lights or electronic billboards. An averaging approach would result in a reconstructed traffic light with a "screen burn-in" artifact: Even when it turns green, the traffic light would still appear to be red in the reconstruction because of all the color evidence gathered from the camera frames waiting for it to turn green.

The fusion per bin approach allows us to limit the growth of the total reconstruction in relation to the image resolution. After the fusion, we backproject all pixels to a new active point cloud in world coordinates, which marks the end of one iteration and allows us to render the points in 3D. For visualizing the active point cloud, we use the same confidence check in the geometry shader to prevent the display of inconfident points. For the confidence check, the parameter T_d can be used to trade-off completeness against accuracy: A higher T_d will allow more but less accurate points, while a lower T_d enforces confident and thus accurate points, but generally fewer points are reconstructed.

5.2.6 Handling of Dynamic Objects

Dynamic objects such as moving vehicles and pedestrians are harder to reconstruct than static objects, *e.g.* house facade. This is due to the more complex correspondences caused by their movement between frames. One would have to detect the object and predict its next position in the next frame or remove it entirely using techniques like semantic image segmentation (see Bârsan *et al.* [Bâr+18]). In a naive 3D reconstruction approach, a dynamic object would appear duplicated because of its changing position. This is called a ghosting artifact. Usually, 3D reconstruction approaches exclude datasets with scenes containing dynamic objects.

We are able to remove ghosting artifacts in the active reconstruction by exploiting the freespace observation of the newest measurement similar to Keller *et al.* [Kel+13]. Parts of the scene that are occluded by a dynamic object may be seen in the new frame again, *i.e.* a new unoccluded depth estimate may exist. This measurement may contradict the reconstructed ghosting artifacts since we are able to look past these objects again, *i.e.* we observe freespace. Now, if a new measurement is farther away than a depth threshold, consisting of the depth of the dynamic object plus an offset in disparity, then it violates our freespace assumption. We remove those closer points immediately, *i.e.* those that belonged to the object that has moved away.



(a) Naive reconstruction. (b) With freespace checks.

FIGURE 5.8: Reconstruction from multiple frames of an intersection scene on KITTI VO, sequence 05. The camera does not move and we use PSMNet [CC18] as the input stereo algorithm.

One example is a pedestrian crossing the frustum from right to left. We display the reconstructed points of the pedestrian as soon as the person enters the frustum. In the next frame, we reconstruct the pedestrian at their new location, but we also still have the reconstructed points of the pedestrian from the previous frame. A well-natured new depth estimate at these ghost pixels will show the background instead, which is why we can remove these ghost pixels. The assumption here is that the stereo matching can differentiate the foreground and background sufficiently, which may not always happen immediately for *e.g.* ELAS. The ghosting artifacts may then exist over multiple frames depending on the chosen stereo matching algorithm. An intersection scene with and without freespace checks is shown in Figure 5.8. Note that the virtual camera is positioned above the vehicle-mounted real stereo camera. With freespace checks, we are able to prevent ghosting artifacts as the car passes the intersection. As a side effect, some point outliers in the scene are also removed, resulting in a cleaner reconstruction.

A limitation of this approach is that we can only remove parts of dynamic objects when we measure the scene behind them again. In a scenario where a vehicle on the neighboring lane moves at a similar speed as the camera and the frustum of the camera cuts the neighboring vehicle, *i.e.* only half of the neighboring vehicle is visible in the current frame, we will see a trailing ghosting artifact chain. This is because the dynamic object is directly transferred to the passive region, which is the region outside the camera frustum. These points are now reconstructed for the passive region in every frame where this dynamic object cuts the camera frustum.

5.3 Mesh-based Reconstruction

In this section, we deal with extending the point-based pipeline to a mesh-based pipeline. Triangle meshes provide a dense surface representation compared to points and do not suffer from the warping / sampling artifacts shown in Figure 5.4. A triangle of a triangle mesh is essentially defining a linearly interpolated area between three points. The handling of these 3D meshes is more complex than handling simple 3D points. Fortunately, we can leverage the modern graphics pipeline to render 3D meshes. Here, we first introduce our depth image triangulation method and how it

can be used for rendering scenes with meshes in our context. Then we integrate this method in the existing point-based framework, while replacing the 1D EIF with the more complex 3D EKF.

5.3.1 Depth Image Triangulation

3D point clouds are a common representation of depth images from a stereo camera. The camera intrinsics and extrinsics plus the depth estimate are used to individually backproject pixels from 2D to 3D. Rendering engines like OpenGL support points as a rendering primitive. Mathematically, points have no area or volume, but they are still rasterized as screen-aligned squares (billboards) with a size parameter. Even though 3D point clouds may appear dense from far distances or with an increased point render size, the representation will thin out when viewed from closer distances. This is due to the sampling and the automatic sizing of the rendered points. The point size can be increased even further, but would eventually reach a point at which the squares are overlapping. On the one hand, one can add a potentially feasible post-processing step in which an explicit interpolation on the rendered image is performed to fill empty pixels. However, this may not be scalable with a large number of empty pixels and might need complex processing logic. On the other hand, implicit interpolation using a triangle mesh can provide a connected surface, which will not thin out compared to points.

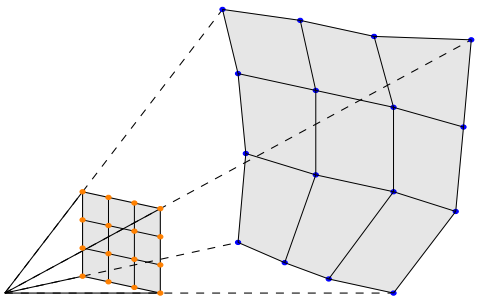


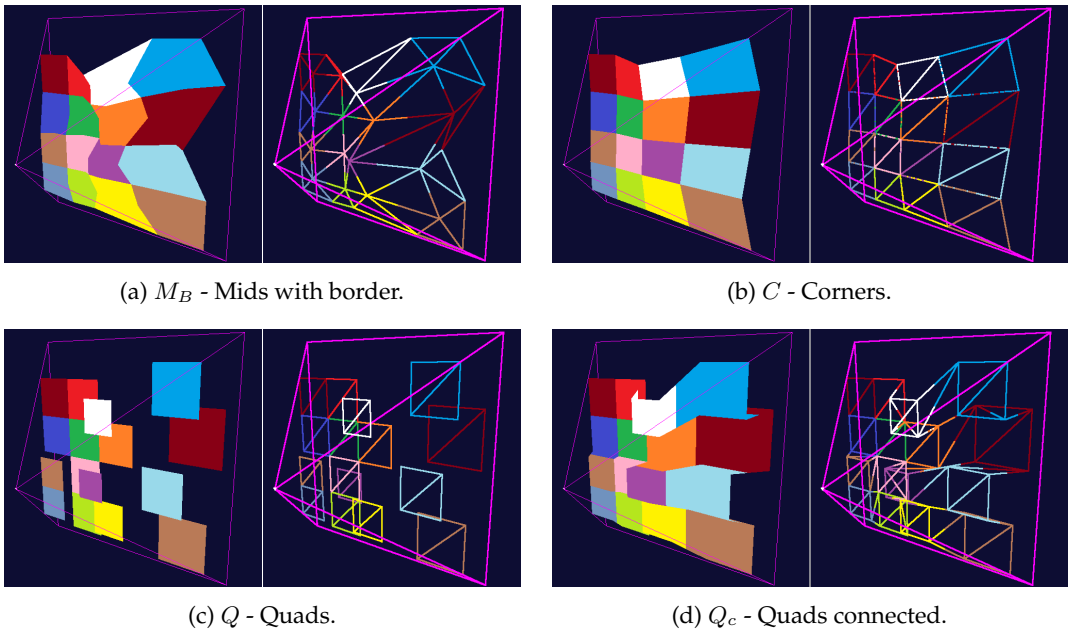
FIGURE 5.9: Schematic illustration of backprojecting pixel corners to 3D according to some distance.

Here, we propose to create a textured 3D mesh directly from the RGB and depth image of a stereo camera. We essentially handle this mesh as a textured 2.5D heightmap as seen from the camera center. Now, the trivial pixel neighborhood can be exploited to transform the image to 3D without connection conflicts or additional processing. Figure 5.9 shows an example variant how a 2D image is displaced to 3D using different depths. Using the pixel neighborhoods is a great advantage compared to 3D reconstruction approaches without such information such as Kazhdan *et al*. [KBH06], which have to work on unstructured 3D data and have to estimate the unknown triangle connections.

5.3.2 Image Triangulation Variants

We propose different ways to triangulate depth images by keeping the vertex count and triangle count to a minimum within certain requirements. Naturally, more variants exists, but may not be necessarily as efficient. We first consider practical aspects of an implementation using shaders in OpenGL.

In contrast to a CPU-based version like Alexiadis *et al*. [AZD13], we render the 2.5D meshes efficiently by offloading computational work to the vertex and geometry shader stages in the rendering pipeline. We precompute the basic geometry of the mesh once on the CPU. We bind both RGB and depth textures to the shader and sample the depth texture according to the UV coordinates in the vertex shader. Then, we can backproject the vertex with the sampled depth to 3D using the intrinsic and extrinsic camera matrix. Afterwards, the vertex is projected onto the virtual camera. In the geometry shader, we can optionally dismiss triangles according to some criteria, *e.g*. triangles that are too large or steep because they bridge the foreground and background. Here, we take the minimum and maximum depth of the triangle vertices in camera space and check if their absolute difference in disparity is smaller than our specified clustering range, *e.g*. $3.0 \cdot e_d$. If this check fails, then we omit this triangle in the geometry shader. This may create holes in the reconstruction, but will also stop triangles from occluding parts of the scene when viewed from a different viewpoint than the camera. In the fragment shader, we sample the RGB texture for the fragment color.



(a) M_B - Mids with border.

(b) C - Corners.

(c) Q - Quads.

(d) Q_c - Quads connected.

FIGURE 5.10: Comparison of meshing variants on the same 4×4 image with different depths and rendered with nearest-neighbor color interpolation for illustration purposes. Left: Solid triangles, right: Wire-frame model.

ID	Name	Vertex count	Triangle count
M	Mids	$w \cdot h$	$2 \cdot w \cdot h - 2 \cdot w - 2 \cdot h + 2$
M_B	Mids with border	$w \cdot h + 2 \cdot w + 2 \cdot h$	$2 \cdot w \cdot h + 2 \cdot w + 2 \cdot h - 2$
C	Corners	$w \cdot h + w + h + 1$	$2 \cdot w \cdot h$
Q	Quads	$4 \cdot w \cdot h$	$2 \cdot w \cdot h$
Q_c	Quads connected	$4 \cdot w \cdot h$	$6 \cdot w \cdot h - 2 \cdot w - 2 \cdot h$

TABLE 5.1: Comparison of different image triangulation variants. w is the image width and h the image height.

In Figure 5.10, we show an overview of all considered triangulation variants and in Table 5.1, we compare the complexity in regard to vertex and triangle count. Note that we have to use two triangles to model the surface of a quadrilateral, *e.g.* a pixel. The first proposed variant M ("Mids") is using pixel center as vertices and connects a pixel center with two neighboring pixel centers per triangle to form a regular grid. Two triangles are used to connect four pixel centers. This straightforward approach misses the outer, half-pixel thick border of the image since we start at the pixel center. To compensate, we add an extra border with triangles that connect the pixel centers to the image border (variant M_B), which we will use in the evaluation for M instead of the borderless variant.

Alternatively, one can use the following variants: The variant C ("Corners") uses the pixel corners as vertices, so that neighboring pixels share vertices. This approach smooths the mesh since the texture sample needs to be interpolated between two or four pixels, but it also means that the original measurement at each pixel is distorted. In other words, the projection of this 3D mesh to the input camera will not be identical to the input depth map.

The variant Q ("Quads") models each pixel individually as a quad. This approach keeps the original pixels intact and it is essentially splatting 3D points with camera-oriented and appropriately scaled quads. One obvious disadvantage of Q is that changing the perspective will reveal the background similarly to a point cloud representation. Variant Q_c ("Quads connected") remedies this problem by adding a connecting extra quad between neighboring pixel edges. This creates a watertight mesh, but almost triples the amount of triangles compared to Q.

5.3.3 Active Mesh Approach

Here, we use depth image triangulation to visualize the current camera viewpoint. We denote this region as active because we can still reobserve scene points in our scenario. Hence, we call it the active mesh.

Our depth image triangulation is essentially a heightmap, but with depths. A heightmap cannot represent objects like tunnels or overhangs in 3D because of its 2.5D nature. To compensate, multiple layers can be used as proposed by Gallup [Gal11], who uses the heightmaps to model the actual height of the scene for 3D

reconstruction instead. In our scenario, we can simplify the use of multiple layers, which we will detail in the following.

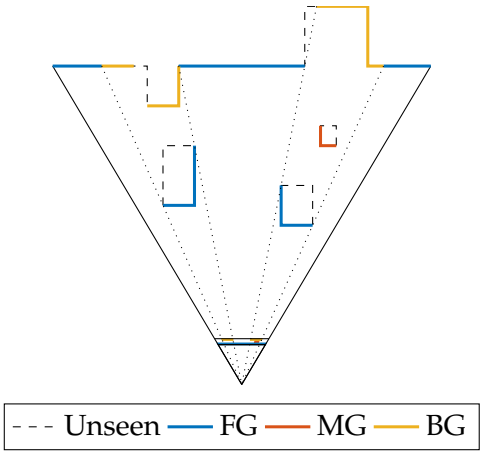


FIGURE 5.11: Top-down view on a reconstructed scene with multiple depth layers.

Figure 5.11 shows a sample reconstruction scenario from a top-down view. Some areas are not seen in the current frame, such as the back of objects (see the dashed lines). Still, we may have seen some other occluded parts of the scene in the previous frames, which means we can still render them. The foreground layer (FG) consists of the nearest geometry, which a ray from the camera center hits. The background layer (BG) consists of the farthest geometry, partially occluded by other layers. Between the FG and BG, there can theoretically be any number of middleground layers (MG), but we only depict one. Note that we have to cut the heightmap at strong depth edges to achieve this reconstruction (see the dotted lines). We only use a single 2.5D mesh for the foreground and also for the background of a scene. The foreground consists of the closest points (like a 3D render), while the background consists of the farthest points. Naturally, this limits the available depth layers to two. However, middleground objects, *e.g.* a lamp post behind a person but in front of vegetation, are not that often encountered in our scenario and potential holes are quickly filled with new observations. Also, the missing objects would have to be occluded, which means that they are not directly seen by the camera anyway. Still, great distances between the virtual camera and the input camera may allow these artifacts to be visible. An additional middleground layer as a straightforward extension will reduce these situations substantially. For now, we will only use two layers in the evaluation.

5.3.4 Tube Mesh Approach

In our scenario with a single front-facing stereo camera mounted on a vehicle, the camera pose movement is dependent on the vehicle movement. Generally, vehicles move mostly forwards, which means camera movement along the optical camera axis. This has the consequence that a new observation of a scene point in a stereo frame will have a smaller depth than the previous observation. This continues until

the scene point is outside the left camera frustum or when there is no correspondence in the right image.

Urban scenarios mainly consist of a U-shaped city canyon, formed by the road on the bottom and house facades on the sides. This allows us to assume that points outside of the current camera frustum will usually not be seen again and we do not have to consider them further for potential fusion. Note that we do not assume a camera with an ultra-wide FoV, which would allow us to see previous points at $90°$ turns.

Still, passed scene points are crucial for visualization because they allow us to synthetically enhance the FoV of the front camera by displaying points from multiple frames. Also, the original FoV of the front camera is smaller than a suitable FoV for immersive use-cases, *e.g.* smaller than $110°$, which is common in VR HMDs.

The last time these newly out-of-view points were seen is in the camera frustum of the previous frame. There, the corresponding scene points were observed with the closest depth over the whole sequence. This means that in the previous frame the corresponding area of a scene object at this point is depicted with the greatest resolution possible. We can exploit this fact for our 3D reconstruction and form a 2.5D mesh out of the previous camera image and pose. In the previous frame, the area of points that are out-of-view in the current frame form a border due to the forwards motion. This border surrounds the points of the previous frame, which are still visible in the current frame. Figure 5.12 shows a schematic situation with three consecutive camera frames. A border in image space can be seen at time t_0 and t_1, colored blue and red respectively, while t_2 (the current frame) has no border yet.

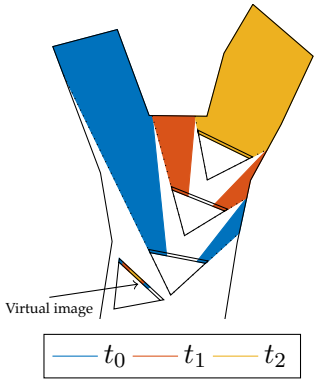


FIGURE 5.12: Top-down representation of the reconstructed area of the tube meshing approach over three frames (t_0, t_1, t_2) and the resulting virtual camera image.

These borders in sequence form a 3D tube over multiple frames, as shown in Figure 5.13, which is the same scene as Figure 1.5. The tube shape with a closed top should only exist in tunnels and the mesh for scenarios with open sky should ideally be U-shaped like the scene. However, due to sky artifacts in the disparity image of real world data the top is also closed. The individual size of the tube elements depends on the camera frame rate and movement.

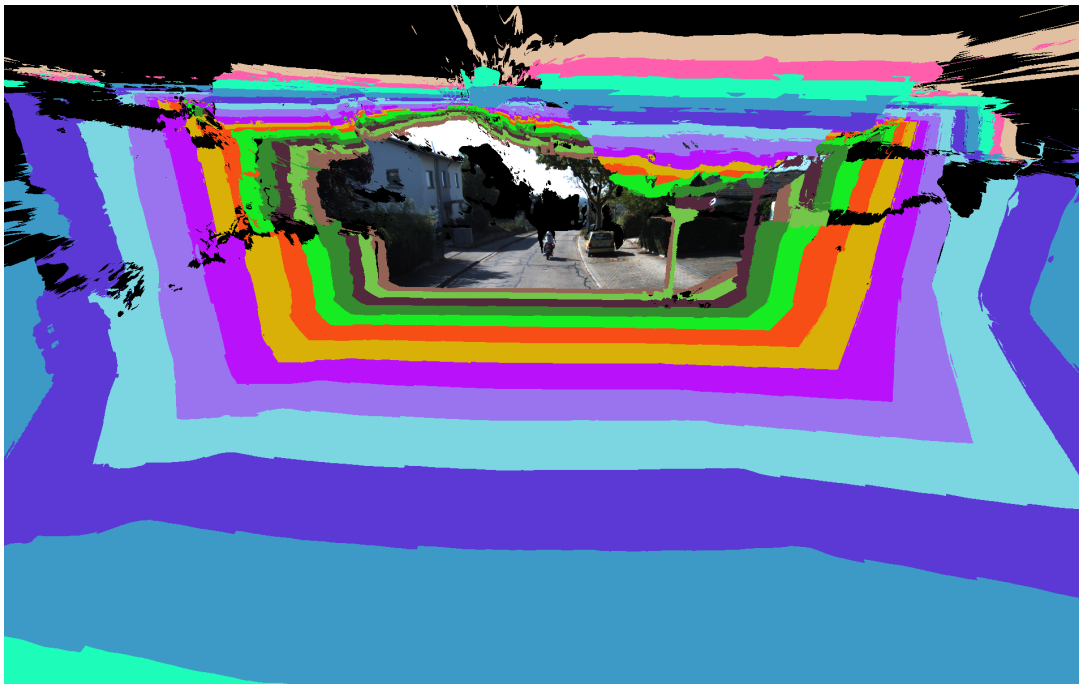


FIGURE 5.13: Reconstruction result of our approach with PSMNet on KITTI VO 02 with visualized tube mesh parts.

To detect the points that constitute the next tube element, we warp pixels of the previous frame to the new camera frame and check if they are out-of-view. We do not disturb the original pixel connections and can extract the relevant parts of the original RGB image and depth map. Each tube mesh element consists of an RGB image, depth image and the associated camera pose, which can be rendered as a 2.5D mesh according to our depth image triangulation.

The elements of the tube mesh are not explicitly connected. The inner border of the current frame can be quite close to the outer border of the next frame when rendered, but they do not overlap in practice or when viewed from angles different than the observation's. We experimented with explicitly closing the tube mesh for a smooth transition via an extra connecting mesh. This small mesh just needs to connect the inner border of one frame with the outer border of the next frame in 3D. This proved to be unnecessarily complex due to foreground objects, which break the usual simple border geometry (see the tree on the right in Figure 5.13 for an example). Our solution is to artificially increase the threshold when a pixel is out-of-view in image space by some amount of pixels (*e.g.* 3 px). Thus, we increase the size of individual tube elements by a certain amount to include some pixels that are still in-view. This enables us to visually close tube gaps very efficiently with little extra cost because we now have small overlapping parts of the individual tube mesh parts.

Besides the front-mounted stereo camera case on a vehicle, one could potentially extend the tube mesh approach to other perspectives. This depends on the scene geometry (*e.g.* distance to house facades), pose movement (*e.g.* maximum yaw angle and length of curves), mounting position (sidewards, centered, front) and camera

FoV. In certain situations, past geometry may be reobserved with the camera, which is not ideal for the tube visualization.

In a front-facing scenario, we can omit parts of the recently reconstructed scene as soon as the vehicle moved far enough because a user is unlikely to turn his head backwards. A backwards facing scenario would require us to flip the tube mesh logic: Points that are new in the current frame form a border around the previous reconstruction. But we still observe this border over potentially lots of frames, which can be used for temporal fusion.

Optimization of Tube Mesh Rendering

The input images for the tube mesh have an exploitable property: They will usually feature a large, centered empty space. This is because the pixels at the border of the image are the ones that are not seen anymore in the next frame due to the camera pose motion along the optical axis. These pixels form a single tube mesh element. The pixels that are going to be reobserved (the empty space in the center) are framed by these pixels. Again, the exact dimensions of these bordering pixels varies each frame and depends on the frame rate and pose movement. Even though we can dismiss triangles that correspond to empty space in a geometry shader or fragment shader eventually, we would still have to call the render function with the maximum triangle count, dictated by the width and height of the image. This will lead to a loss of performance (only up to the geometry shader which can discard empty triangles).

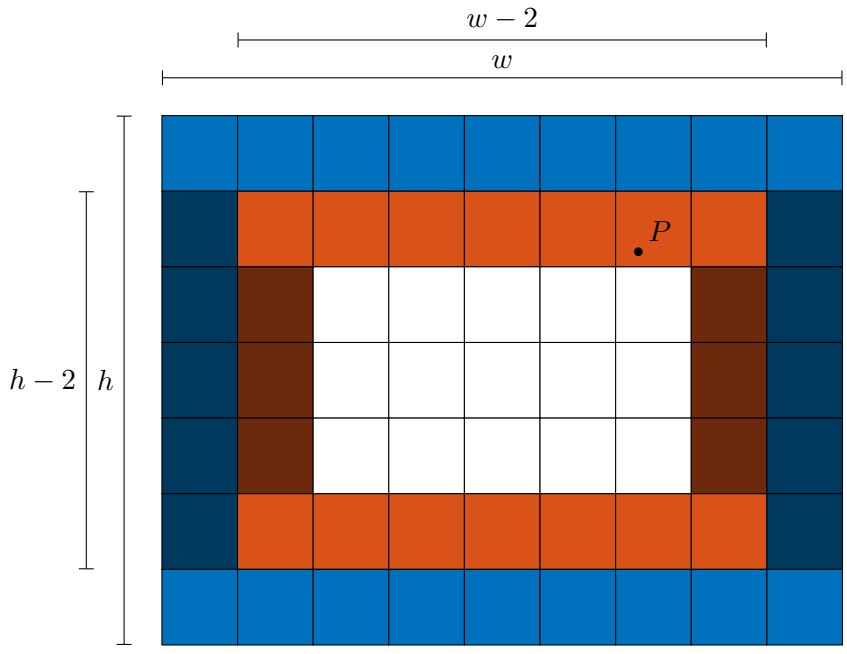


FIGURE 5.14: Illustration of tube mesh optimization in an image with point P, which marks the inner pixel border ($b = 2$). White pixels can be omitted.

Our optimization concept is to rearrange the precomputed triangle sequence in the index buffer, which stores the vertex indices of the three associated 3D points

Iteration	Pixel count
$i = 1$	$2w + 2(h - 2)$
$i = 2$	$2(w - 2) + 2(h - 4)$
$i = 3$	$2(w - 4) + 2(h - 6)$
$\dots$	$\dots$
$i = b - 1$	$2(w - 2(b - 2)) + 2(h - 2(b - 1))$
$i = b$	$2(w - 2(b - 1)) + 2(h - 2b)$
$\dots$	$\dots$
$i = b_{max}$	$\begin{cases} 2(w - 2(b_{max} - 1)) & h \text{ is even} \\ w - 2(b_{max} - 1) + 2(h - 2b_{max}) & h \text{ is odd} \end{cases}$

TABLE 5.2: Number of pixels for the outermost border ($i = 1$) to the inner bordermost b_{max}. w is the image width and h the image height, assuming $w > h$ and $b \geq 5$.

per triangle. Instead of creating triangles left-to-right in a row and row-wise top-to-bottom, we fill the buffer by iterating from the pixel border towards the inside so that each new border forms a $1\,\text{px}$ thick unfilled rectangle outline. For the next iteration, the unfillled rectangle of interest shrinks by two pixels in both horizontal and vertical length. In Figure 5.14, we give an example of the optimization.

The pixel positions for the tube mesh allow us to determine the smallest border that was still hit by a pixel. This inner border b allows us to draw fewer triangles if it is not the innermost border. We do this by computing the associated triangle index of the last element of this border, which we use to render our tube mesh up to this index, resulting in a smaller triangle count to process.

As an alternative, one could devise a spiral-shaped filling of the triangle index buffer. This would allow us to only partially render the innermost border at the cost of more complex index handling. In contrast, we are always rendering up to the last element of the inner border b, even though we may only have hit the first element of it. This is not optimal, but the effect is negligible and simplifies the computation of the last triangle index to render.

We show the number of pixels needed for the pixel border from the outermost to innermost border in Table 5.2 for each iteration. For the outermost border ($i = 1$), we have two pixels rows of length w marked in light blue and two pixel columns of length $(h - 2)$ marked in dark blue as shown in Figure 5.14. As a consequence, the next border will have both their height and width shortened by two. This holds true except for the last iteration. The potential innermost border b_{max} can be computed as follows assuming $w > h$:

$$b_{max} = \left\lceil \frac{h}{2} \right\rceil \tag{5.10}$$

When h is odd, we only have a single row of pixels left. When h is even, it is a double row of pixels. In both cases, w does not play a role in our way of counting. We can generalize the results to get the pixel count for a certain border b (except for

the end edge case b_{max}):

$$i = b \Rightarrow 2(w - 2(b-1)) + 2(h - 2b) = 2w + 2h - 8b + 4 \tag{5.11}$$

Our goal is to determine how many of the specifically arranged triangles we need to render, which is why we compute the total pixel count. Now, to get the total pixel count t_{pc} from the outer border to n, we iterate over all borders $i = 1$ up to b:

$$t_{pc} = \sum_{i=1}^{b} 2w + 2h - 8i + 4 \tag{5.12}$$

$$= b(2w + 2h + 4) \sum_{i=1}^{b} -8i \tag{5.13}$$

$$= b(2w + 2h + 4) - 8 \sum_{i=1}^{b} i \tag{5.14}$$

$$= b(2w + 2h + 4) - 8 \frac{b(b+1)}{2} \tag{5.15}$$

Now, we only have to determine the inner border b to compute the amount of triangles we need to render according to the corresponding total pixel count t_{pc}. We use the minimum of t_{pc} and $w \cdot h$ to handle the b_{max} case, which renders the full resolution in that case.

As an example, the point $\mathbf{P}$ marks the innermost pixel border in Figure 5.14, which means that no pixels are inside the enclosed area of this border. This white area can be safely omitted for rendering. P lies on the second border counting from the outside, so $b = 2$. With $w = 9\,\mathrm{px}$ and $h = 7\,\mathrm{px}$, we can determine t_{pc} to be $48\,\mathrm{px}^2$. This corresponds to ca. $76.2\,\%$ of the total area of $63\,\mathrm{px}^2$ ($w \cdot h$), which means that we can reduce the rendered triangle count by $23.8\,\%$ if a single pixel is rendered via two triangles.

5.3.5 Overview of Mesh-based Pipeline

Figure 5.15 shows an overview of the mesh-based reconstruction pipeline. The pipeline is quite similar to the point-based reconstruction and is run through for each new frame. It consists of an active and passive region. The active region uses the active mesh, while the passive region uses tube meshes for its visualization. For the active region, we iteratively warp the reconstructed geometry to the next frame, in which we get a new RGB and depth image as well as a new camera pose estimate. The warped geometry is binned, clustered and fused analogously to the point-based reconstruction.

One main difference compared to the point-based approach is that we can initialize a 3D EKF for each pixel with a valid depth observation. This means that each point now has to carry a 3×3 covariance matrix instead of a scalar in the 1D EIF case.

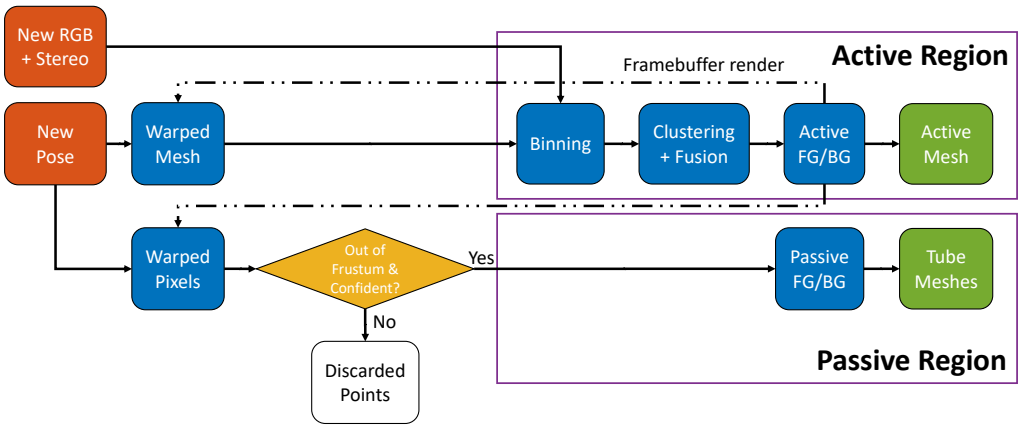


FIGURE 5.15: High-level overview of our mesh-based reconstruction pipeline. Orange represents input, green rendered output and white removal.

The other main difference is that we are now also using meshes. After integrating new measurements in the active region, we can extract the active foreground and background images, which can be displaced as a 3D mesh and directly rendered according to our proposed approach.

To warp these meshes to the new frame, we can render the previous scene reconstruction as seen from the new perspective. For that, we employ an OpenGL framebuffer object, which allows us to render to multiple output textures offscreen. Then, we have to read out the framebuffer object to get another warped foreground and background image of the previous mesh in the new frame. These images can be directly input to the binning stage. However, warping the 3×3 covariance matrices in all pixel bin requires special handling in the pipeline. For that, we encode the covariance matrix as two float textures with three channels. Covariance matrices are symmetric, which means we can omit three of the nine values of the matrix. The three diagonal values of the covariance matrix are saved in the first float texture, while the lower left corner values of the covariance matrix represent the second float texture.

In the warping step of the frame buffer render, we employ an extra optimization. The sequential propagation of all 3D EKF filters on the CPU proved to be a bottleneck. Therefore, we implemented the whole 3D EKF propagation step on the GPU via the fragment shader, which takes all the relevant parameters as input or computes them. In the fragment shader, the covariance matrices are decoded from the two float textures. After propagating the covariance matrix in the fragment shader, we encode it back to two float output textures. Per default, OpenGL uses the *smooth* interpolation qualifier, which is relevant for the encoded covariance matrix and means that the vertex shader output is interpolated in a perspectively correct way across the triangle. The alternative would be to use the *flat* qualifier, which would simply take the value of the provoking vertex of the triangle. We use the *smooth* interpolation qualifier to prevent potential aliasing.

For the passive region, we take the active foreground and background images and warp them to the new frame as described in the previous section for the tube mesh. We discard points that are now in the frustum (with some extra tolerance for tube mesh transitions) and optionally points that are inconfident. Our confidence metric is computed as the root of the trace of the covariance matrix, which allows us to reason about the total variance of a point. This check allows us to discard pixels with a large enough error via a threshold. Then, we determine if this pixel belongs to the foreground or background map. The resulting complete RGB and depth images are added to a a ring buffer of tube mesh visualizations, which have a fixed size in practice to increase performance.

5.4 Dynamic Generation of an Immersive Virtual Sky

In this section, we develop an approach to replace observed sky pixels with a virtual sky. Generally, automotive cameras and other sensors are observing the road and are not pointed up to the sky. Thus, a virtual sky could possibly cover a greater FoV than automotive cameras. For an immersive display of the reconstruction with an HMD, we also have to cover the possibility of the user looking upwards. Without some sort of sky model or background model, the user would only see the uniform background render color. In the following, we discuss why the detection and removal of sky pixel is necessary. We also detail our proposed solution, which uses sky pixels as seed for texture diffusion on a background cubemap.

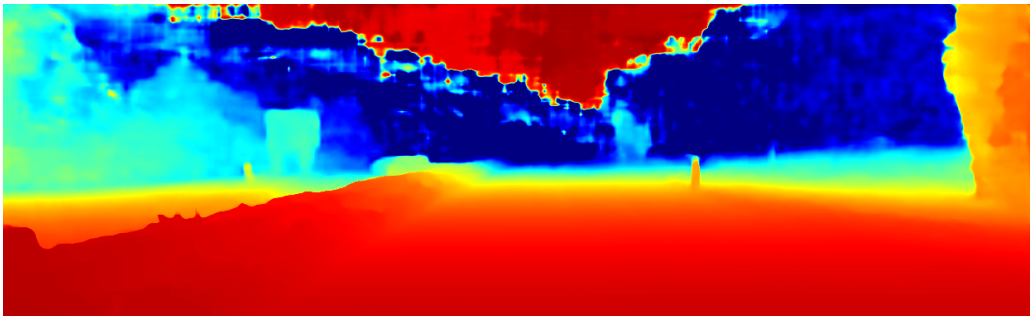
5.4.1 Sky Artifacts

Various kinds of sceneries are challenging for stereo matching algorithms like reflective objects, repetitive textures or homogeneous textures. Homogeneous textures are problematic for stereo matching because they lack visual features to properly find the corresponding scene point in the other image of the stereo pair. In our scenario with a front-mounted stereo camera, we almost always have to deal with the sky as part of the scene that appears to be homogeneously textured. The sky may be uniformly blue or white or can feature clouds. As the sky tends to be bright, the pixels might be oversaturated, *i.e.* the pixels are clipping the color space dependent on the (potentially dynamic) exposure time of the camera. Furthermore, datasets like KITTI use a Lidar ground truth, which cannot measure distances to the sky and mostly faces the road. Machine learning stereo matching algorithms, which are only trained on this dataset or similar datasets, will never have evaluated a sky pixel before and might show undefined behavior for the sky pixel.

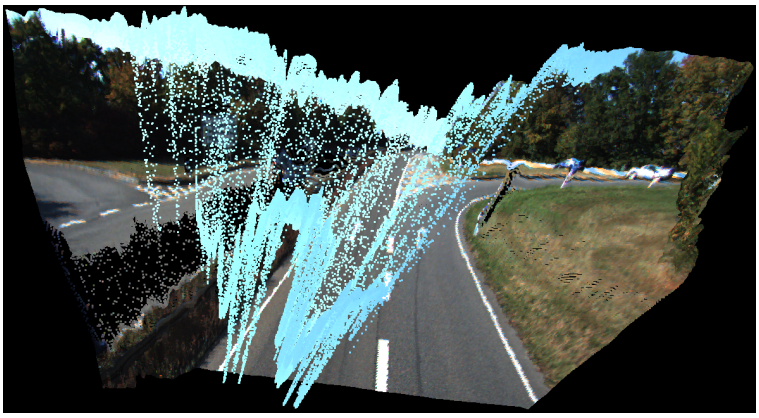
In Figure 5.16, we show a highway scene of KITTI VO using the deep learning stereo method PSMNet with extreme sky artifacts. In the colorized depth image, one can see red or dark-red pixels at the top of the image, which correspond to the sky pixels being as close as the road surface directly in front of the vehicle. These

artifacts become very apparent in the point cloud view, in which spurious sky pixels are occluding the actual scene content. These kinds of artifacts will drastically impact the immersion of users.

(a) RGB image.



(b) Colorized depth image with jet colormap (up to 80 m).



(c) Point cloud view.

FIGURE 5.16: Sky artifacts in a frame of KITTI VO 01 using PSMNet.

Pollefeys *et al*. [Pol+08] proposed to remove parts of the sky by learning its color distribution offline with k-means clustering, considering the most likely sky pixels according to the trained model. However, the simple removal of the sky might not be enough for the immersive case since it will make the empty space in the reconstruction due to the limited FoV even more apparent. Still, removing points also necessarily means detection of the sky pixels, which we could use as input to the generation of a plausible sky.

5.4.2 Skybox Texture Diffusion

Due to the camera FoV, only a certain part of the sky is observed. This detected area can be used as a form of seed for the generation of a virtual sky. Here, our goal is to create an ambient light effect for the sky as a background to accompany the 3D reconstruction, *i.e.* with a skybox. The skybox can be rendered first to create a background for the 3D reconstruction. For the skybox, we detect sky pixels, backproject them to an infinitely far away sphere and diffuse the observed texture of that sphere into the unobserved areas according to their distance to observed pixels. Our implementation is based on the texture diffusion shader concept by DiVerdi [DiV07]. DiVerdi used texture diffusion on a cubemap to fill unobserved pixels in a $360°$ panorama camera sweep. A cubemap is commonly used in 3D rendering and consists of a list of six square 2D textures, which can be used to render a (giant) textured cube seamlessly as a skybox. Each cube face has an associated 2D texture which represents an individual view with a $90°$ view frustum. We can therefore approximate a giant sphere with a giant cube. Here, we also have to react to dynamic changes in the sky over time and cannot guide the camera to unobserved parts of the scene compared to DiVerdi [DiV07].

We determine sky pixels by using semantic segmentation on each image. We employ the deep learning method DeepLabv3 [Che+17] because of its public availability and good segmentation results. For each resulting sky pixel per frame, we set the corresponding depth to $0\,\mathrm{m}$, which masks sky pixels for the actual reconstruction algorithm.

Only the identified sky pixels are then mapped to an infinitely far sphere. This is similar to the 3D video rendering technique of Kang and Shin [KS02], in which they separate the foreground elements from the background and map the static background to a sphere. Figure 5.17 illustrates the mapping of sky pixels. The virtual sky is detected right above the the top of the building and up to the vertical FoV limit. The non-sky objects are not projected onto the sphere. Note that in practice we may still observe some cloud movement depending on *e.g.* the altitude of the observer, wind magnitude and wind direction.

We implement this projection step of the sky pixels by computing the direction of the four corners of the original image with the camera center centered at the origin. We can then use them to render the observed pixels to the cubemap faces, which are hit by the four corners. Cubemaps can be texture sampled in shaders via a 3D direction vector in OpenGL.

Next, we rerender each face of the cubemap with a corresponding camera view. We can then use a texture diffusion algorithm in the fragment shader. Contrary to DiVerdi [DiV07], we do not use an atlas texture to rearrange the cubemap faces in such a way that a smooth interpolation across borders is possible. Instead, we rely on computing 3D directions for the cubemap, which corresponds to the exact pixel centers for sampling. This has the advantage of speeding up the computation, but

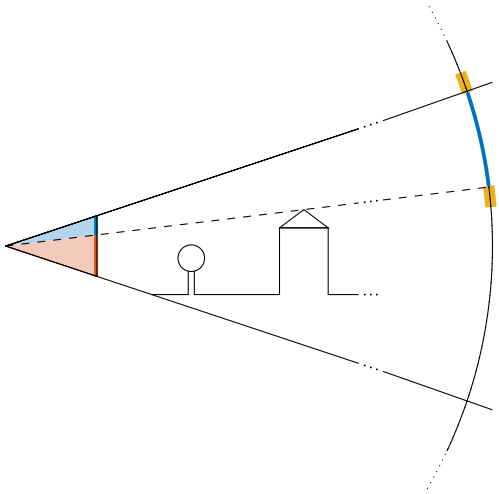


FIGURE 5.17: Side view of a camera in a 3D scene which maps the detected sky pixels in blue to an infinitely far sphere. Yellow areas on the sphere mark the beginning of the skybox texture diffusion.

may have small transition artifacts at the cube face border transitions due to the sequence of computations per cube face.

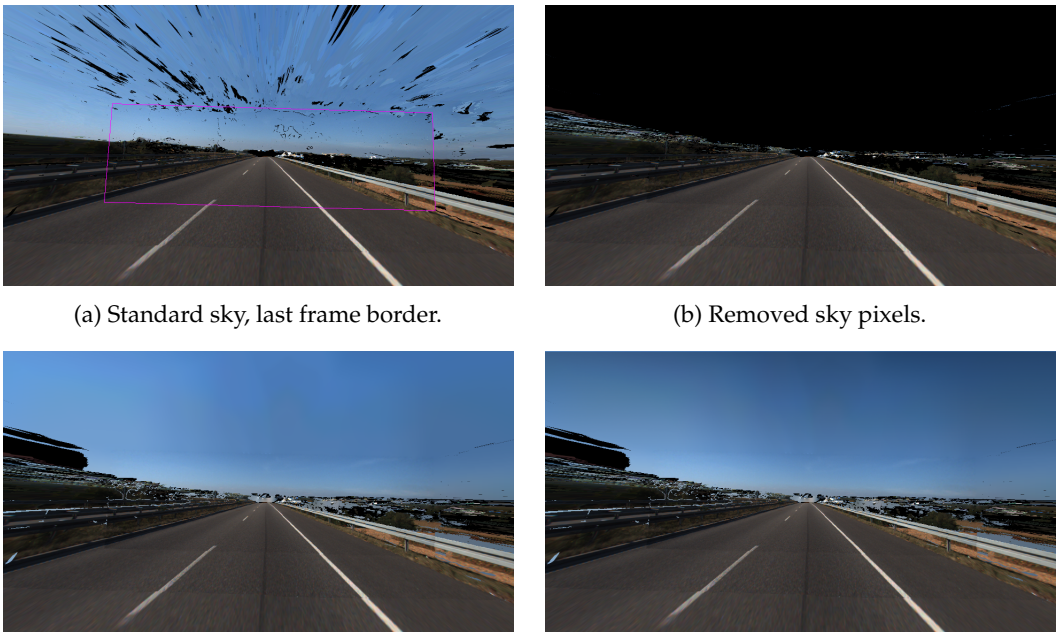
The diffusion algorithm depends on the alpha channel of the cubemap texture. We set an observed pixel to have an alpha of 0, while an unobserved pixel will start at 1. The alpha channel encodes the distance to the nearest observed pixel, transformed to 8 bit (0 to 255).

In Algorithm 1, we depict the basic algorithm on the fragment shader in pseudo-code, builds on top of the algorithm of DiVerdi [DiV07]. First, we initialize an accumulator for samples and a counter. Then, we loop through every pixel in a kernel neighborhood around the center sample. We only accumulate kernel samples when they are closer or have the same distance in the alpha channel compared to the center sample. After the loop, we compute the average kernel and set its alpha channel to the closest observation plus 1 (as integer) since the center sample is another step. Finally, we compare both the center sample with the computed average kernel sample: If the average kernel sample has a smaller alpha, the fragment color will become the average kernel sample. If this is not the case or if we did not find any closer kernel samples, we keep the center sample as the fragment color. Note that we subtract a small constant ϵ, which favors the average kernel sample while acting as a tie-breaker. As a final step, we add configurable decay to the alpha channel of the fragment color to allow for future observations to overwrite previous observations. Without this we would permanently display the directly observed sky at a certain direction, which may include some artifacts or previous and now unfitting observations, *e.g.* the clouds have moved.

We show an example of the original reconstruction, with removed sky pixels and a virtual sky in Figure 5.18 using our mesh-based framework (**3DEKF-F** M with GA-NET, explained in the next chapter) on KITTI VO 01. Note that the deep learning stereo matching GA-NET produces heavy sky artifacts as depicted in Figure 5.16,

Algorithm 1 Texture diffusion pseudo-code in fragment shader.

```
centerSample ← TEXTURE(skybox, centerTexCoords)
accumulatedSamples ← (0, 0, 0, 0)
count ← 0
alphaDecay ← 0.025                    ▷ Decays alpha to be overwritten in the future
minDistance ← 255                     ▷ Controls the reach of the diffusion
for all pixels in kernel except the center do        ▷ Kernel size configurable
    Δ ← COMPUTEPIXELOFFSETDIRECTION()
    kernelSample ← TEXTURE(skybox, centerTexCoords + Δ)
    kernelPixelDistance ← INT(kernelSample.a * 255)
    minDistance ← MIN(minDistance, kernelPixelDistance)
    if kernelSample.a ≤ centerSample.a then      ▷ Closer or same distance
        accumulatedSamples ← accumulatedSamples + kernelSample
        count ← count + 1
    end if
end for
avgValidKernelSample ← accumulatedSamples/FLOAT(count)
avgValidKernelSample.a ← (FLOAT(minDistance) + 1)/255
if count = 0 or centerSample.a ≤ avgValidKernelSample.a − ϵ then
    FragColor ← centerSample
else
    FragColor ← avgValidKernelSample
end if
FragColor.a ← FragColor.a + (1 − FragColor.a) * alphaDecay
```



(a) Standard sky, last frame border.

(b) Removed sky pixels.

(c) Virtual sky via texture diffusion.

(d) Virtual sky with additional gradient.

FIGURE 5.18: Different sky visualizations behind a 3D reconstruction with GA-Net as seen from a position two frames before the current frame.

because we use a camera location near the road. These artifacts are clearly visible in Figure 5.18a as differently colored areas in the sky. Also note that Figure 5.18a

has two visible mesh transitions, which helps to pinpoint the current frame in the image. In Figure 5.18b, we have removed the sky and in Figure 5.18c we use our skybox texture diffusion. The virtual sky looks very consistent and only has a barely noticeable transition from the current frame. Nonetheless, the upper parts of the sky should appear darker in reality due to Rayleigh scattering. As an extension, one could try to mimick different atmospheric effects on top of the skybox texture diffusion. For instance, multiplying a gradient with the upper sky pixels in the cubemap textures to darken the pixels when the camera is looking at the horizon. In Figure 5.18d, we demonstrate how such an extension could look like by applying a gradient.

Chapter 6

3D Reconstruction Results

This chapter evaluates our proposed 3D reconstruction approaches on various datasets in different benchmarks. For that, we analyze quantitative results in comparison to other approaches and also evaluate the reconstructions qualitatively, including a showcase and limitations.

6.1 Benchmark Datasets

First, we introduce the two main datasets in our quantitative evaluation: KITTI VO and SYNTHIA. KITTI VO presents a real car drive through urban scenarios with a sparse Lidar ground truth, while SYNTHIA is a simulated environment using a game engine, featuring a dense ground truth depth.

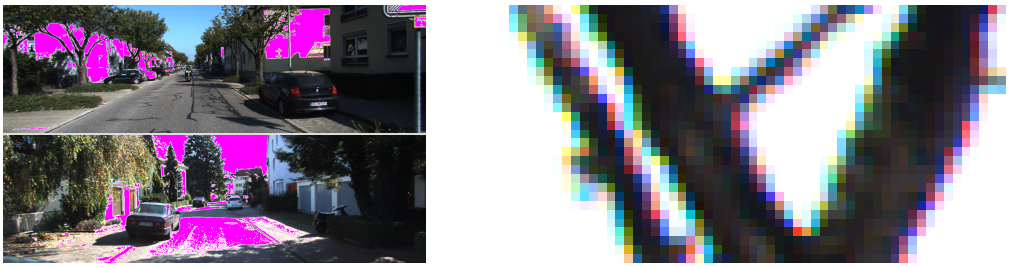
6.1.1 KITTI 2012 (Visual Odometry)

The KITTI 2012 visual odometry [GLU12] dataset's purpose is to evaluate the estimation of the vehicle pose against a ground truth over a sequence of images. For the training sequences, a ground truth pose is provided, as well as two stereo camera pairs and a synchronized $360°$ Lidar. On top of the car, a grey and color camera pair is mounted next to the Lidar. Even though the official purpose of this dataset is the evaluation of VO, this dataset is often used to evaluate 3D reconstruction approaches from a moving vehicle (see [GZS11; ABD13; Sen+13; Tan+16; CBM17; Bâr+18])). It fits our use case quite closely and provides semi-dense ground truth depth estimates via Lidar.

Here, we only use the training dataset because ground truth poses are provided. As mentioned before, we recalibrate the Lidar-to-camera calibration to improve the quantitative evaluation for subpixel accuracy (see Haeling *et al*. [HNS19b]). Unfortunately, there are no raw calibration parameters provided for sequence 03 (and the corresponding recording day). We exclude this sequence from the evaluation for this reason. We also only focus on the color camera stereo pair. The remaining dataset consists in total of $22\,390$ frames of RGB image pairs. A showcase of the different sceneries in the KITTI VO dataset is shown in Figure 6.1. A rectified KITTI image is around 1250×350 pixels large and images are taken at a frame rate of $10\,\mathrm{Hz}$, which corresponds to ca. 37 minutes of total driving time. The stereo baseline is ca. $0.54\,\mathrm{m}$.

FIGURE 6.1: Images from the KITTI odometry dataset featuring urban roads, suburban areas, roads through forests and highways.

Compared to modern standards, the video resolution is rather low and the baseline compared to in-series automotive stereo cameras is more than twice as long. This means that the baseline in KITTI has positive consequences for the depth accuracy, but the low resolution has negative consequences because it impacts the focal length in pixels.



(a) Pure white pixels of same sequence colored in magenta.

(b) Chromatic abberation at high contrasts. Tree branches depicted of top left image.

FIGURE 6.2: Color clipping and chromatic abberation artifacts observed in all sequences of KITTI VO.

We observed two relevant artifacts in the KITTI VO image sequences, which we illustrate in Figure 6.2. First, the shutter time of the cameras is adjusted dynamically, but in some images the exposure time is too long. Figure 6.2a shows pure white pixels recolored in magenta of two images of the same sequence that are ca. 13 s apart after a right turn. Note that the sky is bright blue in the top image and completely white (*i.e.* magenta) in the bottom image. A pure white pixel is evidence that the color space is clipped, *i.e.* the pixel cannot be any brighter because of the limited dynamic range of the camera. For traditional stereo algorithms that focus on a local area, these pixels are impossible to match since these regions are all homogeneously textured. For example, ELAS fails in these areas.

A second artifact can be seen at high contrast areas, *e.g.* edges of a tree in front of a white facade as seen in Figure 6.2b. There are strongly miscolored pixels at the tree branch. False color tones from red to blue are apparent on right edges, while the left edges feature false color tones from blue to green. We hypothesize that this may be caused by chromatic abberation, which is a distortion induced by the camera

lens. Different wavelengths of light can have slightly different effective focal lengths, which can explain this artifact. We hypothesize that this effect might make it more difficult for stereo matching to find great matches in these images since the other camera needs to have very similar artifacts.

6.1.2 SYNTHIA

The second dataset we use is the SYNTHIA dataset [Ros+16]. It consists of photo-realistic scenes rendered from a moving vehicle in the game engine Unity. There are season and weather variations available for SYNTHIA. We chose the SUMMER sequences as they provide a clean and bright view like the KITTI dataset. Two omni-cameras with a large baseline of $0.8\,\mathrm{m}$ take eight synchronous images, each omni-camera consisting of four individual cameras looking to the front, left, right and back direction of the vehicle. Figure 6.3 showcases different scenes as seen from the left front camera. We use both front-facing cameras as a stereo camera plus the provided poses for our stereo reconstruction. The frame rate of SYNTHIA is $5\,\mathrm{Hz}$ and a frame has a resolution of 1280×720 pixels. The frame rate correponds to only one half of the frame rate of KITTI VO, but there are more than twice as many pixels in the image. Hence, both datasets use roughly the same amount of pixels in the same timeframe.

FIGURE 6.3: Sample images from the SYNTHIA dataset featuring highways, urban roads, tunnels and pedestrians.

One advantage of this synthetic dataset is that dense depth ground truth is available, *i.e.* a depth for each pixel of each image. This enables additional evaluations that can even reason about the structure of the projected scene. Unfortunately, the rendered trees in the scene consist of multiple billboards for the highly detailed leaves, which may provide implausible ground truth depths at actually transparent objects. Billboarding is a 3D graphics rendering technique, which consists of efficiently rendering screen-space aligned textured quads instead of highly detailed geometry.

Figure 6.4 shows a close-up of this artifact. The billboards are appear coarse in the ground truth depth image compared to the RGB image.

FIGURE 6.4: Ground truth depth artifacts caused by billboarded trees.
Left: RGB image, right: Ground truth depth image.

Another point to mention is that the scene was rendered with slight anti-aliasing, while the depth has sharp boundaries as expected from the Z-buffer output. This means that there are already ghosting artifacts present when rendering the RGB image with the associated ground truth depths. Transparent foreground elements will be seen in the background due to the aforementioned billboarding and they form a slight silhouette of the foreground in the background.

Furthermore, the virtual camera does not remain still when *e.g.* stopping the vehicle. There exists a slight camera shake, *i.e.* slight variations in the camera pose, which may be a way to mimic the effects of slight engine vibrations. An advantage of SYNTHIA is that we also have different views available of the scene from the same point in time. For instance, we use both side views of the the left omni-camera to evaluate the whole 3D reconstruction that uses only the front stereo camera pair. For that, we only use the RGB images to propose a challenging view prediction scenario. Note that the sidewards-facing cameras may see directly into areas of the stereo shadow, *i.e.* areas that are occluded for the front stereo camera.

6.2 Reconstructor Overview

We compare both our point-based and mesh-based reconstruction frameworks to naive approaches and other approaches from the literature. We refer to each 3D reconstruction approach as "reconstructors". As input, we use both a conventional stereo method and a deep learning method, which we will detail in the following. In this chapter, we will highlight all reconstructor names in bold for clarity.

As our first input method, we use ELAS as a conventional stereo matching result, which is also evaluated as raw stereo for each frame. We use the default "ROBOTICS" setting of ELAS for all datasets.

For deep learning stereo matching, we use the accompanying pre-trained KITTI 2012 stereo models. We also compare these raw stereo matching results to other reconstruction approaches. Analyzing the KITTI 2012 stereo training dataset has shown that only four unique calibration files have been used. Three of the four unique calibration files are identical to the VO training dataset sequences (00-02, 03, 04-10), totalling in 187 out of 193 images. Also, identical images from KITTI 2012 and VO can be found. We therefore conclude that we can safely apply the KITTI 2012 model to the VO dataset without great generalization challenges for the deep learning stereo approaches.

Early tests with publicly available PSMNet and its pre-trained KITTI 2012 models at that time have shown that the disparity error distribution is biased on KITTI 2012 stereo, KITTI 2015 stereo and KITTI VO in our predicted images. Fusion approaches cannot improve the accuracy in this situation, but only the precision. This is why we omit PSMNet from the quantitative evaluation, but still use it for qualitative purposes as it shows consistent surfaces and provides a depth estimate for each pixel.

To accommodate, we use GA-Net and its pre-trained KITTI 2012 model ("**Raw GA-Net**"), which has excellent results on KITTI 2012 in our tests, contrary to PSMNet. Even though the computed recalibration of the Lidar to camera transformation with ELAS improves the raw results with GA-Net on KITTI VO, we apply a further individual recalibration for GA-Net to reduce the bias even more. It is possible that other effects like camera calibration errors may occlude the true source of the observed bias, *i.e.* which stereo matching disparity error is not inherently centered at 0. This is why we use two different offsets for the recalibration. Still, the evaluation on SYNTHIA reduces this sort of bias, but introduces a generalization problem, *i.e.* the photo-realistic synthetic context. Using the KITTI 2012 model with GA-Net on down-sampled SYNTHIA images proved to be not competitive in terms of stereo accuracy. Still, we could observe that the stereo matching is less porous than ELAS. For SYNTHIA, we could not refine the GA-NET model or use the full resolution due to hardware limitations.

TABLE 6.1: Overview of evaluated 3D reconstructors.

Name	Fusion	Reconstruction
Raw ELAS [GRU10]	-	Current frame raw stereo
Raw GA-Net [Zha+19]	-	Current frame raw stereo
Union	-	Union of all stereo measurements in 3D
Replace	Replace with new	Warping and replace
SS [GZS11]	50/50	Warping and thresholded fusion
ALC [ABD13]	Error-weighted	Warping, geom. and photom. consistency
CEIF	1D EIF	See Section 5.2
3DEKF	3D EKF	See Section 5.3

Table 6.1 shows an overview of the evaluated approaches. We use the originally proposed parameter set for each algorithm. "**Raw ELAS**" and "**Raw GA-Net**" denote the raw stereo matching input to our reconstructors and form a baseline in image

space accuracy. Note that these two methods are only reconstructing the single current frame, which makes them unviable for actual 3D reconstruction purposes. "**Union**" is the combination of each stereo 3D measurement up to this point, while "**Replace**" warps pixels to the new frame and replaces old pixels with the newest depth measurement. We reimplemented the approaches of StereoScan [GZS11] ("**SS**") and Alcantarilla *et al*. [ABD13] ("**ALC**"), which both use the KITTI dataset and ELAS as input. For **ALC**, we do not consider future frames in the reconstruction because all the other approaches can work online or incremental. For **SS**, we disregard unfused points in the 3D reconstruction and only show fused points, which showed great quality improvements.

In the following, we explain the parameter set of our proposed approaches. The 1D point-based reconstructor using 1D extended information filters is denoted as "**CEIF**" (clustered EIF), while the 3D mesh-based reconstructor using 3D extended Kalman filters is called "**3DEKF**". For both, we use a start stereo matching error of $\sigma_d = 0.7\,\text{px}$. For σ_d, we computed an empirical error distribution of ELAS on all KITTI sequences and estimated a rough stereo matching error. The confidence threshold parameter is different between both approaches because the 3D approach has two additional dimensions (with higher accuracy). For **CEIF**, we used a confidence threshold T_d of $0.65\,\text{px}$ and $0.7\,\text{px}$ for **3DEKF**. The confidence thresholds are low enough to only show fused (and accurate) points. In Subsection 4.6.2 we discussed the limitations of this particular confidence thresholding. **3DEKF-C** denotes the confidence-thresholded variant, while **3DEKF-F** uses no confidence-thresholding. Both use the M meshing variant as described in Subsection 5.3.1 when not mentioned otherwise. Both use a clustering threshold of $T_c = 3.0$, as described before.

The start pointing errors for the **3DEKF** are $\sigma_u = \sigma_v = 0.25\,\text{px}$, which are plausible values for an accurate camera calibration reprojection error. We set the translational pose error (standard deviation) in each dimension to $5\,\text{cm}$ and the angular error to $0.01\,\text{rad}$, while **CEIF** also uses $5\,\text{cm}$ for the translational pose error. This roughly corresponds to the average achievable visual odometry results in the same dataset (see Geiger *et al*. [GLU12]).

For SYNTHIA, there is a slight perspective change and a lower frame rate compared to KITTI. We increase the confidence threshold slightly to $0.45\,\text{px}$ for **CEIF** and $0.8\,\text{px}$ for **3DEKF** to still visualize very near depths of the road ahead (see Subsection 4.6.2). Since the camera pose is also highly accurate if not perfect, we can increase our accuracy estimate for the pose errors. We adjust the translational pose error for **CEIF** and **3DEKF** to $0.05\,\text{cm}$. We use $5\,\text{px}$ extra border for the transition of the tube meshes in the **3DEKF**.

Again, we applied the KITTI 2012 model to GA-Net on down-scaled SYNTHIA images due to our limited available GPU memory, but the results were not comparable to ELAS. Training GA-Net on SYNTHIA was also not possible due to hardware restrictions, which is why we omit a quantitative evaluation of GA-Net on SYNTHIA.

6.3 Quantitative Evaluation

We are using three different approaches for the quantitative evaluation of 3D reconstructions, which all work in image space by comparing the reconstructed image to a ground truth image. First, we use standard stereo metrics, which measure the per-pixel geometrical error of depth images. Second, we use the HCI stereo metrics (Heidelberg Collaboratory for Image Processing, see Honauer [Hon19]) to compare the quality of different geometrical structures in the reconstructed depth image such as surfaces or thin structures. Third, we try to predict camera views by rendering the reconstruction from a certain camera view and comparing the render to the corresponding original RGB image, resulting in a quantifiable view prediction error. Now, also color information matters and not only depth or normal information. In the following, we first explain our reasons for this method, explain the used metrics, and present and discuss the results subsequently. Furthermore, we also discuss time performance of each reconstructor and analyze how our reconstruction algorithm may intentionally be degraded in an ablation study.

6.3.1 Method

We evaluate the 3D reconstruction approaches in the image space of the current frame, proposed by both Bârsan [Bâr+18] and Sengupta *et al*. [Sen+13]. For that, the current reconstruction is projected to the current frame and compared to the ground truth pixels. Note that for KITTI, we do not have a dense ground truth available, but we can project the Lidar measurements to image space. The projection implies that the nearest point along a pixel ray volume will become the projected pixel.

Alternatively, Tanner *et al*. [Tan+16] aggregated all KITTI Lidar points in a giant point cloud, so that every reconstructed 3D point can be matched to the closest ground truth point. This approach gives a good visual impression of the error locations, *e.g*. far away points on the road ahead before a turn. We evaluated this approach with *nanoflann* [Bla14], which allows fast nearest-neighbor searches with large point clouds. We also removed reconstructed points per frame that lie outside the FoV of the Lidar scanner because it mostly observes points near ground level. Our results showed that the error metrics of this approach correlate highly with the depth error in the image space evaluation. Furthermore, it is very resource intensive and not natively usable with meshes as it relies on point clouds as input. For this reason, we only evaluate in the reconstructions in image space.

For practical purposes, we limit the size of the reconstructed point cloud to 20 times the size of one KITTI frame. We use a looping pool of 3D points, which will be replaced. With this, we can avoid the related computational costs of keeping every previously reconstructed frame in memory. Higher values did not change the results of the evaluation significantly. The only exceptions are the **Union** approach, which uses 100 times the size of a single frame KITTI frame and the **Raw Elas / Raw GA-Net** approach, which uses the size of a single frame. We also limit the

reconstructions of KITTI to $50\,\mathrm{m}$ and for SYNTHIA to $60\,\mathrm{m}$ to suppress strong outliers for all reconstructors.

6.3.2 Error Metrics

Here, we present the used error metrics in image space for our benchmarks. The stereo metrics deal with the geometrical error, while the HCI stereo metrics reason about the geometrical structure such as surfaces and the view prediction error represents comparisons of rendered RGB images of the reconstruction with the ground truth RGB images of different camera perspectives.

Stereo Matching Error Metrics

For the stereo error metrics, we compute both disparity and depth error. For each, we report both the average RMSE (root-mean-square error) and the average median, which is less affected by outliers. We evaluated other metrics such as bad pixels (used in Middlebury and KITTI) and error quantiles, but these metrics provide a very similar ranking of the reconstructors. Therefore, we omit them for brevity. Furthermore, we provide the average depth median as a metric to compare the depth distributions. This helps us to see if an algorithm deviates strongly from the input stereo baseline, *e.g.* by only reconstructing points with near depths. To compare the precision of the estimates, *i.e.* how the error itself varies over a sequence of frames, we compute the standard devation for both disparity RMSE and the depth RMSE. This allows us to reason about the stability of the reconstruction, which needs to be consistent over multiple frames.

We also employ the KITTI stereo metrics from Menze *et al.* [MG15], which consist of two parts. One is accuracy, which is the number of accurate pixels divided by the total amount of pixels. Here, a pixel is classified as accurate if the disparity error compared to the ground truth is less than $3\,\mathrm{px}$ or smaller than $5\,\%$ of the true disparity. The second term is called completeness. It is the ratio of the number of ground truth pixels that are "hit" by a corresponding reconstructed pixel over all ground truth pixels.

Additionally, we introduce a new metric for the depth error that we call normalized depth error. Its motivation is to introduce a difficulty term for the associated actual depth of a pixel. For instance, close points are expected to be better reconstructed in terms of their depth than points farther away. With this, we can even compare incomplete reconstruction results (with empty pixels) more directly. For a single pixel with reconstructed depth z and associated ground truth z_{gt}, the normalized depth error is computed as follows:

$$\text{Normalized Depth Error} = \frac{|z_{gt} - z|}{e_{z_{gt}}} \tag{6.1}$$

$e_{z_{gt}}$ is computed from the stereo depth error model of Equation 4.1 with ground truth depth z_{gt} as input. We normalize the absolute depth error by the expected depth error from the model. The other parameters are the baseline B, focal length f and the stereo matching error e_d, which we set to $0.7\,\mathrm{px}$ throughout this evaluation. An observed absolute depth error of $0.4\,\mathrm{m}$ at a ground truth depth $z_{gt} = 20\,\mathrm{m}$ ($\rightarrow e_{z_{gt}} = 0.71\,\mathrm{m}$) with KITTI parameters ($B = 0.54\,\mathrm{m}$ and $f = 700\,\mathrm{px}$) has a normalized depth error of 0.56. The same absolute depth error at a far depth of $z_{gt} = 30\,\mathrm{m}$ ($\rightarrow e_{z_{gt}} = 1.58\,\mathrm{m}$) would only be 0.25 and at a near depth $z_{gt} = 10\,\mathrm{m}$ ($\rightarrow e_{z_{gt}} = 0.18\,\mathrm{m}$) it would be 2.20 because of the relative size of the absolute error.

HCI Stereo Metrics

The HCI stereo metrics as proposed by Honauer [Hon19] are new performance metrics for depth estimation algorithms, which go beyond per pixel evaluation: The structure of the depth image is estimated, which allows reasoning about discontinuities, continuous surfaces and fine structures. The motivation behind these extra metrics is the observation that stereo matching algorithms perform differently on different geometric entities. This means that maybe the best performing stereo matching algorithm in a benchmark may not be the best for each particular use case. Some example use cases include: A needle during surgery needs to always be fully reconstructed (fine structures), occlusion of virtual objects in AR requires exact object borders in the real scene (discontinuities) or a 3D scanned object needs to be appropriately shaded and illuminated (continuous surfaces).

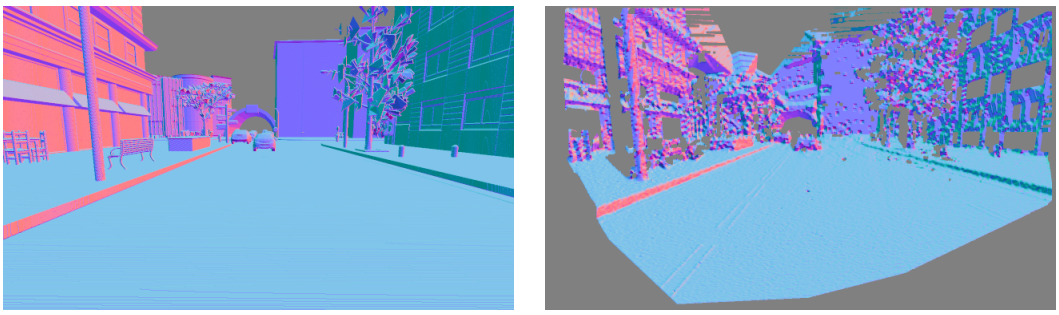
We implemented the metrics according to Honauer [Hon19], which we will describe in the following. We highlight the names of the HCI stereo metrics in bold. Note again that we need a dense ground truth to compute these metrics, which means that in our case we can only compute them on synthetic data in practice because of the availability of dense ground truth data for whole image sequences.

(a) Original depth. (b) Extrapolated FG. (c) Extrapolated BG.

FIGURE 6.5: Colored ground truth depth of a bench with extrapolated foreground and background.

For depth discontinuities, we compute the **foreground fattening** and **foreground thinning** metric. For that, we compute high gradient pixels of the reference depth image and compute an extrapolated foreground and background map by extrapolating nearby pixels from the gradient in the normal direction either with the depth of the FG or BG. Figure 6.5 shows an example of the resulting images. In case of the extrapolated FG, the outline of the park bench grows, while in the extrapolated BG

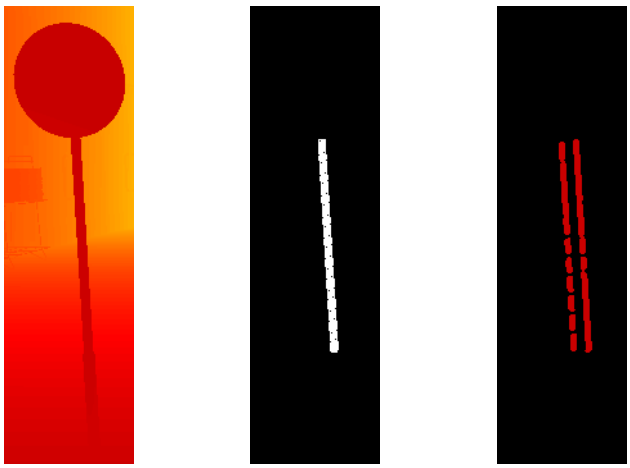
case, the fence-like structure is replaced with the background. After computing these images on the ground truth, we can quantify whether the reconstructed pixels in this region are closer in depth to the ground truth or to the respective extrapolated map. For example, **FG fattening** would thus be the percentage of pixels that are closer in depth to the fake extrapolated FG than the ground truth in the high gradient image region. In Figure 6.5, that would correspond to pixels that are closer to the fake outline of the park bench instead of the actual ground truth background.



(a) Ground truth. (b) ELAS.

FIGURE 6.6: Colored normal maps in camera space of the ground truth and ELAS.

For continuous surfaces, we can detect these regions by computing the second derivative on the reference disparity image and identify pixels close to 0. For these specific regions, we can compute the **angular error** per identified pixel on computed normal maps in camera space (see Harms *et al.* [Har+14]) as shown in Figure 6.6. **Bumpiness** and **smoothing** require an estimate of the local curvature at a pixel. If the magnitude of the curvature at a reconstructed pixel is greater than the reference, then a pixel is considered bumpy with an associated value. If it is smaller, then it is smoother.



(a) Original depth. (b) Fine structure. (c) Extrapolated FG.

FIGURE 6.7: Detecting fine structures (traffic sign pole) on the colored ground truth depth and extrapolating the FG.

For fine structures, the region is defined by shifting ground truth pixels along

their signed gradient and identify the small regions with overlap. Figure 6.7 shows an example of a traffic sign. The pole of the sign is detected as a fine structure (middle image) and we also show the extrapolated FG of the masked fine structure (right image). **Detail fattening** is analogue to foreground fattening for this region. The other metric **porosity** quantifies how close pixels of the reconstructed regions are to the nearest fine structure. Lastly, **Fragmentation** quantifies how many 8-connected components there are in the fine structure regions of the ground truth compared to the reconstruction.

View Prediction Error Metrics

In the view prediction error metrics, we want to evaluate whether geometrical error improvements can also be quantified in visual improvements in a view prediction error benchmark. Now, the color information also matters, which is not the case for the stereo metrics and the HCI stereo metrics.

The additional camera views of SYNTHIA allow us to compare the reconstruction of the side views, which were reconstructed using only the front cameras. This enables us to evaluate the visual quality of the reconstructed images quantitatively in a similar vein to video view interpolation methods (see Zitnick *et al*. [Zit+] or Waechter *et al*. [Wae+17]). There, the virtual images from a multi-view stereo setup are compared to the original images of an individual camera. We use three view prediction error metrics for this. The first one is the structural similarity index [Wan+04] (SSIM, here: mean SSIM of color channels), which aims to be a perceptual metric rather than a purely signal-based one. The second one is the proposed 1-NCC error metric from Waechter *et al*. [Wae+17]. This metric compares local patches according to their normalized cross correlation (NCC). Their implementation includes a small numerical offset for computation of the NCC to compensate for low-textured regions, *i.e*. regions with low local variance, which we also employ. Note that SSIM does take empty pixels into account, whereas we skip empty pixels for the 1-NCC error. We explicitly count missing pixels in our third metric, completeness. This metric is similar to the completeness from the stereo metric context, but here it will always assume a $100\,\%$ complete ground truth image.

6.3.3 Stereo Matching Error Results

Table 6.2 depicts the average stereo benchmark results over all KITTI VO frames. Our confidence-thresholded **3DEKF** dominates in the disparity error metrics, and is the second place in depth error metrics after **ALC**. This is only due to depth selection, which we will address later in this section. Both standard deviations σ of the RMSE in disparity and depth over all frames are significantly lower than the input and all other approaches, which confirms the hypothesized improvement in precision from temporal fusion. Additionally, our full **3DEKF** approach is able to extend the average completeness by 13.5 percentage points compared to **Raw ELAS**, while having a

lower error than **Union**. **CEIF** tends to perform relatively well in the ranking, but is outperformed in terms of disparity and depth error by a significant amount by **3DEKF-C**. This observation holds true for **CEIF** in all the following stereo matching error evaluations.

	Disparity error in px			Depth error in m			Depth in m	KITTI Metrics		Seconds
Name	**RMSE**	σ	**Median**	**RMSE**	σ	**Median**	**Median**	**Acc.**	**Compl.**	**Time / frame**
Raw ELAS	4.453	2.636	0.440	2.336	0.907	0.167	11.800	0.909	0.778	0.009
Union	9.602	7.794	1.536	3.759	1.501	0.399	10.361	0.759	**0.948**	**0.007**
Replace	5.468	3.292	0.485	2.618	1.012	0.173	11.363	0.888	0.882	0.091
ALC	6.073	5.532	0.828	**1.699**	1.252	**0.150**	8.388	0.894	0.403	0.507
SS	6.212	4.977	0.797	2.491	1.167	0.224	10.462	0.875	0.639	0.044
CEIF	3.858	2.383	0.456	2.378	1.027	0.189	12.328	0.929	0.735	0.241
3DEKF-C	**2.859**	**1.521**	**0.436**	1.970	**0.771**	0.176	12.285	**0.946**	0.681	0.218
3DEKF-F	5.695	3.709	0.531	2.430	0.960	0.179	10.989	0.892	0.913	0.197

TABLE 6.2: Stereo benchmark on KITTI VO with ELAS. Results depict the mean over all frames.

For GA-Net as stereo matching algorithm on KITTI VO in Table 6.3, the results are quite similar except that the input has a lower median error than all other approaches. The median might not matter that much for the visual quality, but strong outliers, that heavily influence the RMSE, might. Note that we do not reach 100 % completeness because we omit very far depths.

	Disparity error in px			Depth error in m			Depth in m	KITTI Metrics		Seconds
Name	**RMSE**	σ	**Median**	**RMSE**	σ	**Median**	**Median**	**Acc.**	**Compl.**	**Time / frame**
Raw GA-Net	5.565	4.067	**0.362**	2.305	0.877	**0.116**	10.970	0.919	**0.999**	0.011
Union	10.599	8.783	1.849	4.058	1.575	0.484	10.080	0.736	**0.999**	**0.009**
Replace	5.657	4.064	0.364	2.343	0.879	0.116	10.966	0.879	**0.999**	0.105
ALC	7.714	6.540	0.879	2.104	1.374	0.159	8.325	0.877	0.518	0.676
SS	6.773	5.289	0.843	2.583	1.187	0.234	10.476	0.871	0.796	0.049
CEIF	4.277	2.713	0.386	2.233	0.833	0.134	11.352	0.935	0.921	0.260
3DEKF-C	**3.341**	**1.943**	0.389	**2.057**	**0.728**	0.135	11.553	**0.948**	0.883	0.216
3DEKF-F	5.423	3.994	0.420	2.233	0.841	0.132	10.951	0.926	0.989	0.230

TABLE 6.3: Stereo benchmark on KITTI VO with GA-Net. Results depict the mean over all frames.

Comparing **Raw ELAS** in Table 6.2 with **Raw GA-Net** in Table 6.3, we see that **Raw GA-Net** has a higher completeness and a lower depth median. Generally, **Raw ELAS** constructs a Delaunay Triangulation with feature points, which will omit pixels at the border of the image if no feature is found in this area as illustrated in Figure 6.8. This means that potentially close points in an urban scene are omitted, *e.g.* the road and near house facades, which explains why **Raw GA-Net** has a lower depth median overall.

	Disparity error in px			Depth error in m			Depth in m	KITTI Metrics		Seconds
Name	**RMSE**	σ	**Median**	**RMSE**	σ	**Median**	**Median**	**Acc.**	**Compl.**	**Time / frame**
Raw ELAS	6.357	6.432	**0.852**	2.011	1.029	0.221	6.080	0.948	0.772	0.020
Union	16.860	15.910	0.973	3.407	1.464	0.057	5.607	0.882	0.890	**0.015**
Replace	9.530	8.881	0.898	2.628	1.404	0.047	5.965	0.934	0.865	0.189
ALC	11.349	27.105	1.278	**1.512**	1.327	**0.033**	4.017	0.932	0.511	1.231
SS	13.872	25.942	0.960	2.951	1.604	0.063	6.492	0.907	0.561	0.090
CEIF	8.630	7.960	0.872	2.570	1.330	0.046	6.097	0.939	0.789	0.491
3DEKF-C	**6.029**	**6.228**	0.885	1.880	**0.997**	0.048	6.026	**0.952**	0.757	0.397
3DEKF-F	8.448	7.613	1.000	2.139	1.089	0.040	5.219	0.942	**0.934**	0.397

TABLE 6.4: Stereo benchmark on SYNTHIA with ELAS. Results depict the mean over all frames.

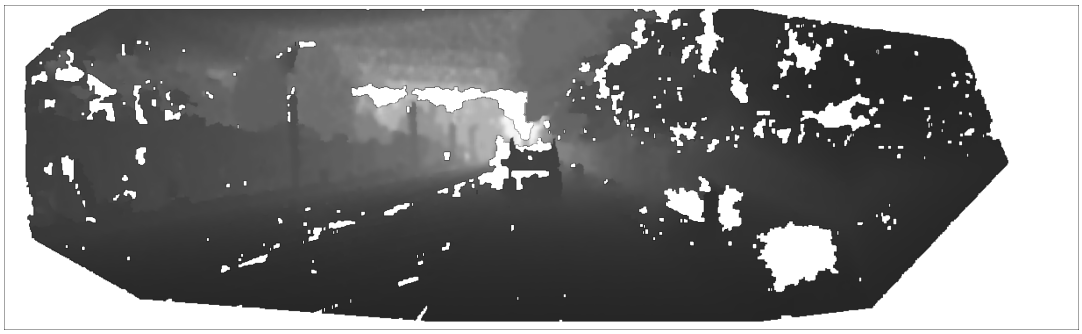


FIGURE 6.8: Typical depth image of ELAS on KITTI VO 04 in the form of a convex polygon.

Additionally, we show quite similar stereo results on SYNTHIA with ELAS in Table 6.4. Here, our full **3DEKF-F** approach gains 15 percentage points in completeness compared to the input, which is similar to the result of the KITTI dataset. We outperform even **Union** because the camera in SYNTHIA shows more of the street surface up-close than KITTI, which makes sampling artifacts more prevalent. We assume that this is because the KITTI images are cropped on the lower part of the images to hide the hood, while SYNTHIA has no visible vehicle chassis and thus no need for cropping.

Comparing the full **3DEKF-F** approach to the point reconstructor **Replace** on both datasets, we surprisingly find similar results. Since **Replace** only takes ca. half the time per frame, it might be worthwhile to consider **Replace** as an easy-to-implement candidate for a practical implementation or to see if a simple mesh-based version of **Replace** could be even better. Here, **3DEKF-F** has a really good depth error median due to an apparent preference for near points as seen in the depth median.

Comparing the results of our **3DEKF** in the confidence-thresholded version (**3DEKF-C**) with the version without it (**3DEKF-F**), we see that we might tradeoff geometric quality against reconstruction completeness. For instance, in immersive applications we might prefer to avoid holes, *i.e.* we prefer completeness over accuracy.

In Figure 6.9, we show a comparison of the KITTI accuracy and completeness term with a $1 \cdot \sigma$ covariance contour as 2D ellipses. The orange **Union** ellipse represents an upper limit for the completeness of the point reconstructors on ELAS, but it performs worst in accuracy due to near outliers that win the depth buffer test and occlude other points behind it. Also note the wide range of **ALC** and **SS** in terms of accuracy. Compared to the **Raw ELAS**, our **3DEKF-C** approach has a smaller accuracy variance but higher completeness variance, and the **3DEKF-F** has a smaller completeness variance and a slightly higher accuracy variance.

Furthermore, we compare the accuracy of the input stereo against our **3DEKF-C** approach over all individual sequences on KITTI VO. In Figure 6.10, we show the corresponding boxplots, both for ELAS and GA-Net as input. There exists no single sequence in which we do not improve the mean accuracy. Additionally, the visualized variance after fusion is much smaller compared to the input. We therefore

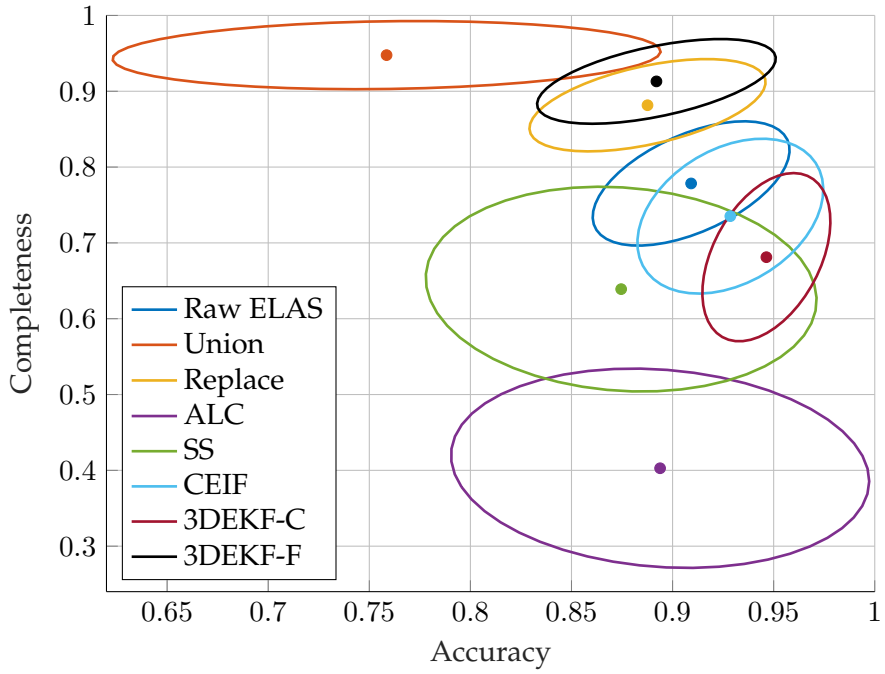


FIGURE 6.9: Comparison of average accuracy vs. average completeness of 3D reconstructors over all KITTI VO frames.

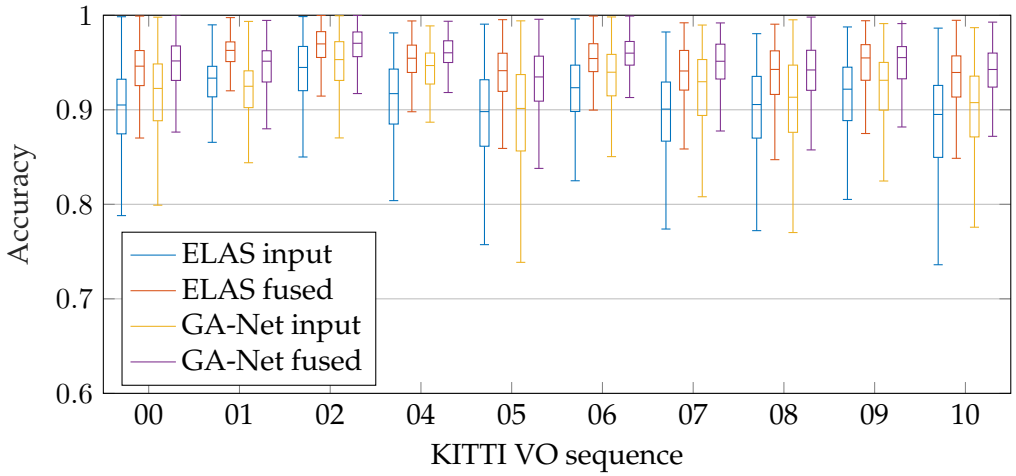


FIGURE 6.10: Boxplots for accuracy over all KITTI VO frames.

can increase both accuracy and precision with our **3DEKF-C** approach.

In the following subsections, we evaluate the effects of depth selection and the effects of different options in our pipeline.

Depth Selection

Our confidence-thresholded **3DEKF-C** approach fares well in terms of disparity, but seems to be lacking in terms of depth accuracy compared to **ALC** on KITTI. We argue that this is due to depth selection, *i.e.* **ALC** is more likely to reconstruct near geometry which has a lower stereo error. This is because **ALC** uses a geometric and photometric threshold between frames, which heavily favors near depths. In Figure 6.11, we show

a depth histogram over all KITTI VO frames with ELAS as input with 1 m bins. **ALC** concentrates more on near points close to 10 m than all other approaches. In fact, no depth beyond 17 m was reconstructed, which also explains its very low completeness score. Depth selection thus can trade off disparity accuracy for depth accuracy.

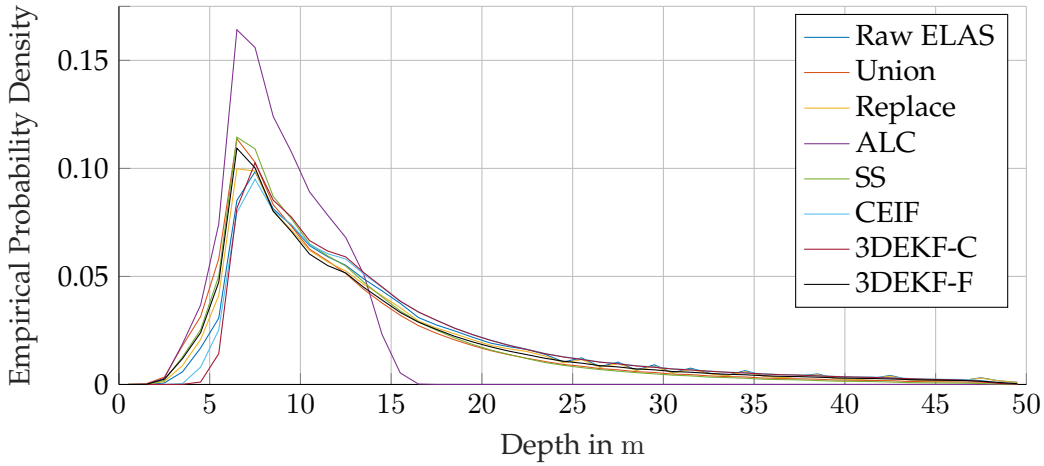


FIGURE 6.11: Probability density function of all recorded depths over all KITTI VO frames, binned to 1 m.

Note that in Figure 6.11 **Raw ELAS** shows small peaks in the graph at higher depths, which is evidence for disparity pixel-locking. Pixel-locking in stereo matching means that the stereo matching is more likely to output integer disparity values instead of sub-pixel accurate estimates. For instance, a disparity of 8 px corresponds to ca. 48 m and 7 px to ca. 43 m with KITTI stereo parameters, which both are recognizable in Figure 6.11. The same graph for GA-Net looks very similar, except that **Raw GA-Net** is a little shifted to the left since ELAS will only reconstruct the inner part of an image. This is again because of its Delaunay triangulation, which tends to not include the image border as vertices. This image border may include very near scenery at the bottom of the image. Also, GA-Net features no apparent pixel-locking.

Arguing that a method is biased towards near depths may also be interpreted in the way that our reference method is biased towards far depths, *i.e.* our reconstruction favors low disparity values. This is partly true: The median depth for **3DEKF-C** is ca. 0.6 m higher than the input in Table 6.2. Comparing **3DEKF-C** with **3DEKF-F** (and from visual inspection), we see that the confidence-based method is lacking very near points from 3 m to 7 m, which can be very accurately reconstructed in depth. We argue that this is an effect of non-linear disparity sampling as described previously in Subsection 4.6.2. As an alternative, marking all points below 7 m as confident worsened both the disparity and depth error, but improved the completeness. Similarly, lowering our confidence threshold had the similar effect.

In Figure 6.12, the median of all disparity median errors per frame over all KITTI VO frames in relation to the ground truth disparity is shown. For that, we computed a disparity median error per ground truth disparity bin (0 px to 100 px, 1 px step) for each frame. Note that we skipped empty bins of a frame when computing the median

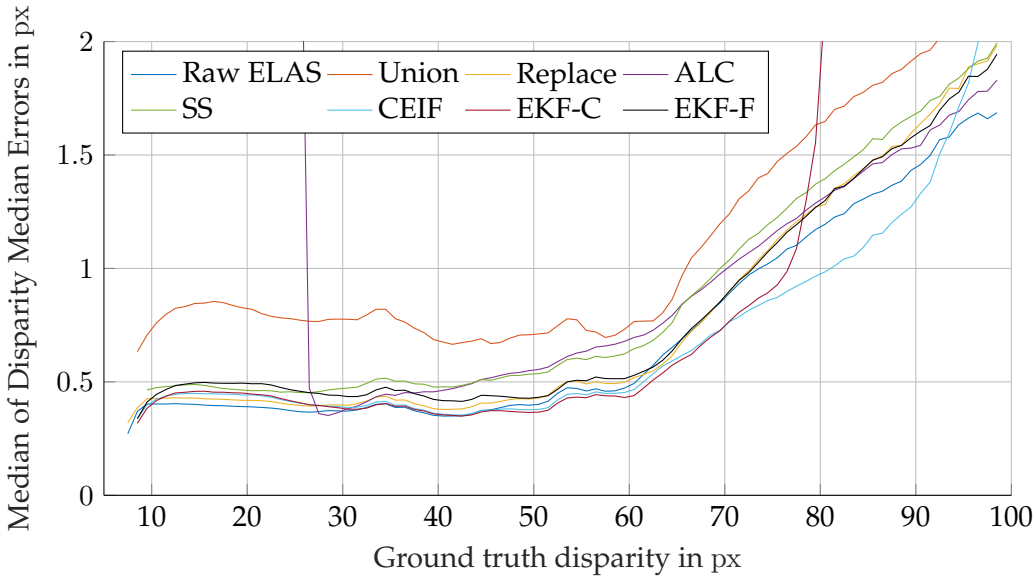


FIGURE 6.12: Median of disparity median errors over all KITTI VO frames with ELAS over the ground truth disparity in $1\,\mathrm{px}$ bins.

over all frames. **ALC** and **3DEKF-C** both have sudden jumps in the error. For **ALC** this is due to the fact that there exist barely any reconstructed points below $26\,\mathrm{px}$ disparity, *i.e.* $> 14.5\,\mathrm{m}$ (see Figure 6.11). The **3DEKF-C** begins to deteriorate at ca. $70\,\mathrm{px}$ disparity, *i.e.* $< 5.4\,\mathrm{m}$, which also correlates with the lack of samples as shown in Figure 6.11.

The main bulk of the points is at $7.5\,\mathrm{m}$ depth for all approaches, which corresponds to a disparity of ca. $50\,\mathrm{px}$. Here, the **3DEKF-C** has an advantage compared to all other approaches in the median disparity error. At lower disparities / higher depths, the **3DEKF-C** may lose its advantage due to more difficult correspondences between frames, *i.e.* angular camera pose errors that scale with distance. **Raw ELAS** generally outperforms ALC, which questions the excellent depth reconstruction results of **ALC**. Furthermore, all median errors higher than $60\,\mathrm{px}$ ($< 6.3\,\mathrm{m}$) disparity are increasing. This can again be due to the decreasing availability of samples or because of the calibration from the ground truth Lidar sensors to the camera pair, which could induce depth-dependent errors by an inaccurate translational offset. This inaccurate offset becomes an increasingly greater part in relation to the decreasing depth. The analogue chart for SYNTHIA does not show this characteristic, but it shows a linearly increasing error with the ground truth disparity itself.

Our proposed normalized depth error metric is the ratio of the absolute depth error and the expected depth error given an estimate of the stereo matching error e_d. This metric over all frames is shown as boxplots in Figure 6.13. Here, we can see that **ALC** is not performing as well in the depth error as before. Our **3DEKF-C** shows the best performance with a lower median and lower variance than competing approaches. Thus, by weighing the absolute depth error with its estimated difficulty to achieve a fair comparison of depths, our reconstruction approach comes out on

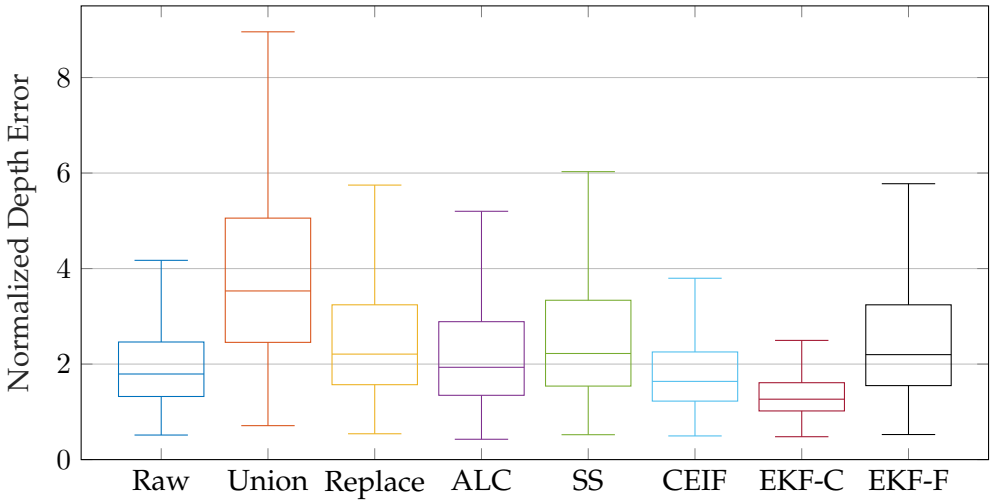


FIGURE 6.13: Normalized depth error boxplots over all KITTI VO sequence frames using ELAS as input.

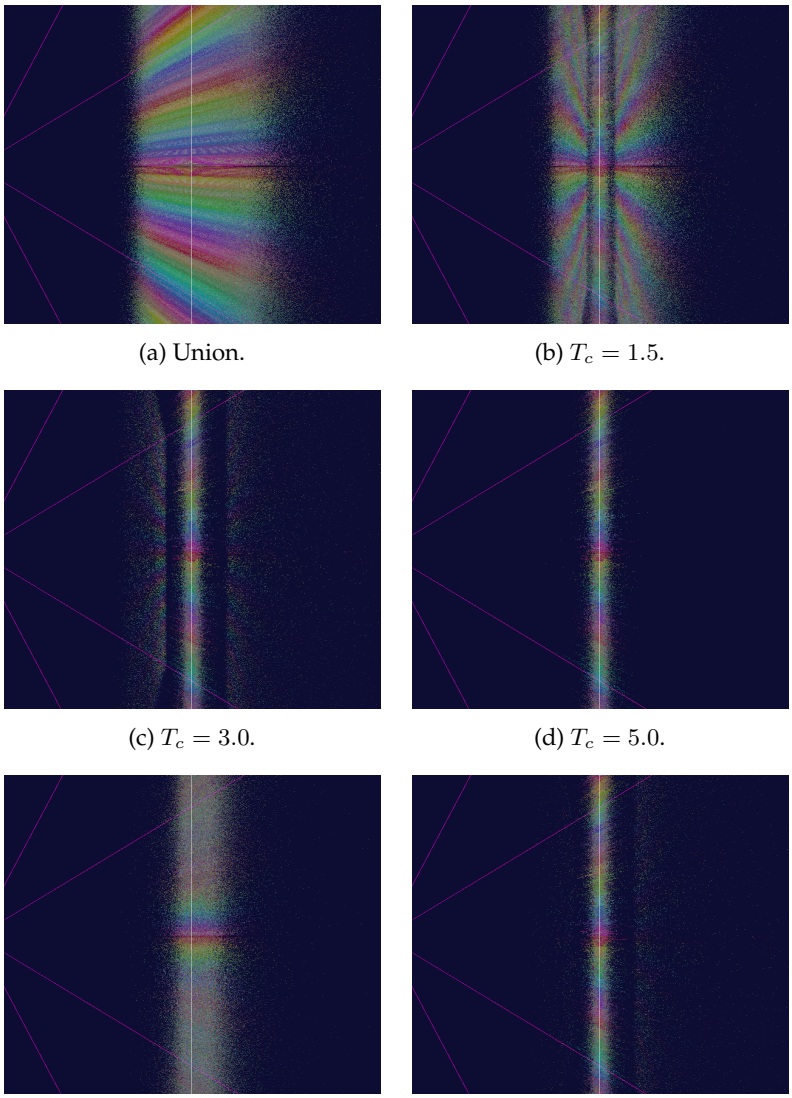
top.

Effects of Clustering, Binning and Freespace Checks

Here, we analyze and explore the interaction of clustering, binning and freespace checks on synthetic and real data using our point-based reconstructor **CEIF**. First, we look at the effects of changing the cluster threshold T_c of the Z-test and toggling the freespace check on a static, synthetic scene. Second, we evaluate the effects of different binning choices and the freespace check on KITTI VO using ELAS. Only the binning component does not apply to the **3DEKF** aswell.

To evaluate the effects of different clustering parameters, we create a synthetic scene, which consists of a $1280\,\mathrm{px} \times 720\,\mathrm{px}$ test image wall seen from $50\,\mathrm{m}$ away. The camera does not move and we use 50 frames of the scene with a synthetic disparity error on the ground truth of $e_d = 0.5\,\mathrm{px}$ using a normal distribution. The stereo camera has a focal length of $f = 700\,\mathrm{px}$ with a baseline of $B = 0.54\,\mathrm{m}$, which is similar to KITTI VO. We use the same parameters for **CEIF** as described in Section 6.2, unless explicitly noted otherwise. The disparity error e_d has changed, which is why we also change the confidence threshold to $T_d = 0.25$. This behaves similarly to the original parameter set. Additionally, we toggle the freespace check.

In Figure 6.14, we show different reconstructions of the wall after 50 frames from a top-down view with a perspective camera. Note that we draw the ground truth wall in white and that the camera center lies to the left of the view. Also note that the normally distributed disparity error at such a high depth will result in a skew towards higher depths. Figure 6.14a shows the **Union** of all input points, which need to be fused and clustered to a single depth layer. Figure 6.14b, Figure 6.14c and Figure 6.14c show the effects of different clustering thresholds T_c and no freespace checks. For $T_c = 1.5$ and $T_c = 3.0$, there exist two clearly visible additional depth layers forming outside the actual wall. These ghost walls exists because over 50 frames there are



(a) Union. (b) $T_c = 1.5$.

(c) $T_c = 3.0$. (d) $T_c = 5.0$.

(e) Without leeway FSC, $T_c = 3.0$. (f) With FSC, $T_c = 3.0$.

FIGURE 6.14: Static, synthetic wall observed with CEIF over 50 frames, shown from a top-down view. Results shown for various clustering options.

enough observations outside the clustering range to form additional confident depth layers. For very small clustering thresholds T_c, there could exist even more ghost walls. The $T_c = 5.0$ version does not have this problem because samples outside the $5 \cdot \sigma$ Z-test are very rare. But in practice this leeway may lead to the fusion of actually separated objects along their depth, *i.e.* false correspondences.

In Figure 6.14e and Figure 6.14f, we use both freespace checks (now: FSC) and $T_c = 3.0$, but Figure 6.14e does not use the extra leeway of $T_c \cdot e_d$ pixels disparity before removing a point which occludes another point. This results in a very random looking color distribution. Figure 6.14f represents our standard parameter set and shows reduced outliers compared to the variant without FSC in Figure 6.14c. For practical purposes, a clustering threshold of $T_c = 3.0$ with the full leeway FSC represents a good tradeoff between reducing ghost layers and separating different

objects along the depth. Only a small second layer is observed behind the ground truth, which, in practice, is occluded from the reconstruction on the front. Here, in this synthetic example using normally distributed noise, spatial correlations may improve or alleviate some of the observed issues, *e.g.* Total Global Variation as demonstrated by Tanner *et al.* [Tan+16].

Next, we evaluate the effects of the binning choice and freespace check for our point-based reconstruction. We evaluate four different binning variants for **CEIF** and whether our freespace check helps. The variants are nearest neighbor (NN), inverse distance weighting (IDW), sticky inverse distance weighting using points outside the bins (SIDW) and the pixel size weighting approach (PSW). If not otherwise specified, **CEIF** uses the default parameters on KITTI VO.

CEIF variant		Disparity error in px			Depth error in m			Depth in m	KITTI Metrics		Seconds
Name	**FSC?**	**RMSE**	σ	**Median**	**RMSE**	σ	**Median**	**Median**	**Acc.**	**Compl.**	**Time / frame**
NN	No	2.930	1.660	0.432	2.530	0.916	0.272	15.324	0.926	0.558	0.174
NN	Yes	2.758	1.674	0.413	2.247	0.892	0.258	15.350	0.941	0.545	**0.165**
IDW	No	2.667	1.355	0.433	2.293	0.819	0.234	14.208	0.929	0.767	0.239
IDW	Yes	2.533	1.353	0.407	2.079	0.814	0.219	14.231	**0.944**	0.752	0.231
SIDW	No	2.666	**1.345**	0.432	2.293	0.812	0.233	14.195	0.929	0.763	0.225
SIDW	Yes	**2.529**	1.349	**0.406**	**2.076**	**0.811**	**0.218**	14.224	**0.944**	0.749	0.215
PSW	No	3.011	1.656	0.465	2.517	0.868	0.248	14.020	0.917	**0.795**	0.429
PSW	Yes	2.784	1.690	0.423	2.137	0.851	0.226	14.092	0.940	0.776	0.398

TABLE 6.5: Stereo benchmark of CEIF with different options on KITTI VO sequence 04 using ELAS.

Table 6.5 shows the evaluation of the different **CEIF** variants on KITTI VO sequence 04 using ELAS. This sequence features 271 frames of a straight forward movement, some incoming traffic and a trailing vehicle in front of the camera with some distance. Comparing the variants with FSC and without it, we can see that it plays a significant role in improving the disparity and depth error for every single variant. The accuracy also improves, but we may lose some pixels in the reconstruction since we remove parts of the scene.

The NN variant delivers the fastest results, but with relatively bad stereo error and low completeness. Using less pixels for the reconstruction can obviously improve the reconstruction speed, but heavily fragments the reconstruction due to the nature of point sampling. SIDW has a small edge compared to IDW. SIDW with FSC presents the best option in terms of disparity and depth error overall, which is why we chose this variant in the other evaluations for **CEIF**. PSW fares similar to NN in terms of disparity and depth error, but is able to prevent the aforementioned fragmentation of point sampling, resulting in the highest relative completeness. This unfortunately comes at a performance cost because it needs roughly twice as much time to integrate a new frame into the reconstruction compared to NN.

6.3.4 HCI Stereo Results

We show the results of the average values for the HCI stereo metrics in Table 6.6 with ELAS as stereo matching input on all SYNTHIA frames. Note that we computed the mean of the thin structures metrics without the frames that feature no detected thin structures (*e.g.* no lamp posts in that frame of the sequence). Here, lower values mean

better performance. First, there exists no clear dominating reconstructor, which is also the exact motivation for the HCI stereo metrics. Second, even though a clear improvement of our approaches can not be seen compared to the **Raw ELAS** baseline, we still have very similar results to the input. Thus, our approach does not deteriorate the respective input quality in terms of these metrics, which is a positive trait. Third, point reconstructors may have subpar results because of sampling holes. For instance, **ALC**'s porosity is extremely high for this reason.

	Discontinuities		Surfaces			Thin Structures		
Name	**FG Fattening**	**FG Thinning**	**Ang. Error**	**Bumpiness**	**Smoothing**	**Porosity**	**D. Fattening**	**Fragmentation**
Raw ELAS	36.186	13.091	0.316	32.333	0.871	178.090	45.855	**67.130**
Union	69.388	**6.280**	0.688	415.310	0.440	60.949	66.740	73.443
Replace	37.798	15.019	0.340	61.672	0.856	**69.046**	44.486	75.595
ALC	63.482	7.909	0.335	219.240	0.545	1713.600	68.281	85.159
SS	44.403	15.651	0.620	1069.300	**0.272**	149.630	51.089	87.536
CEIF	**36.278**	14.117	**0.301**	106.310	0.909	157.130	**43.065**	76.506
3DEKF-C	37.215	12.572	0.305	32.112	1.011	186.29	44.306	72.782
3DEKF-F	39.031	13.890	0.316	**25.339**	1.196	89.066	43.654	74.339

TABLE 6.6: Mean HCI stereo metric results over each SYNTHIA frame with ELAS.

Nevertheless, **3DEKF-F** manages to close holes of **Raw Elas** input as seen in the porosity metric. It also scores lower in bumpiness.

6.3.5 View Prediction Error Results

Figure 6.15 shows an example frame of the view prediction error evaluation. The reconstructed images, as well as the ground truth and a visualization of the 1-NCC error are depicted. In the side views, the error at a scene point increases with increased distance to the last real observation from the front stereo camera. We assume that this is because of the increasingly different angles from which that scene point is depicted compared to the last real observation. In practice, the accumulated pose error should also have a similar "aging" effect. Plus, with some distance traveled, unobserved areas of the scenes may now be required to be reconstructed.

	Front			Left			Right		
Name	**SSIM**	**1-NCC**	**Compl.**	**SSIM**	**1-NCC**	**Compl.**	**SSIM**	**1-NCC**	**Compl.**
Raw ELAS	0.634	**0.018**	0.669	0.036	0.531	0.055	0.063	0.439	0.094
Union	0.460	0.502	**0.837**	0.153	0.775	**0.610**	0.198	0.727	**0.660**
Replace	0.671	0.091	0.775	0.102	0.748	0.383	0.158	0.680	0.473
ALC	0.165	0.676	0.411	0.056	0.773	0.207	0.093	0.736	0.302
SS	0.207	0.658	0.469	0.073	0.747	0.253	0.097	0.730	0.316
CEIF	0.636	0.137	0.778	0.090	0.736	0.314	0.150	0.691	0.460
3DEKF-C	0.612	0.036	0.667	0.178	0.474	0.281	0.281	0.422	0.468
3DEKF-F	**0.770**	0.031	0.833	**0.353**	**0.463**	0.525	**0.400**	**0.413**	0.571

TABLE 6.7: Average view prediction errors and completeness of different reconstruction approaches on SYNTHIA with ELAS.

The average view prediction error on SYNTHIA for the front, left and right view of the left omni-camera over all sequences using only the front camera pair as stereo input is shown in Table 6.7. Generally, the **3DEKF-F** with the M meshing variant achieves a high average completeness for each view, only being outperformed by **Union**. **Union** again aggregates all input pixels over multiple frames. **3DEKF-F** also has the lowest average 1-NCC error for the side views and the greatest average SSIM for all views. The front view with **Raw ELAS** has the best 1-NCC error. This is not

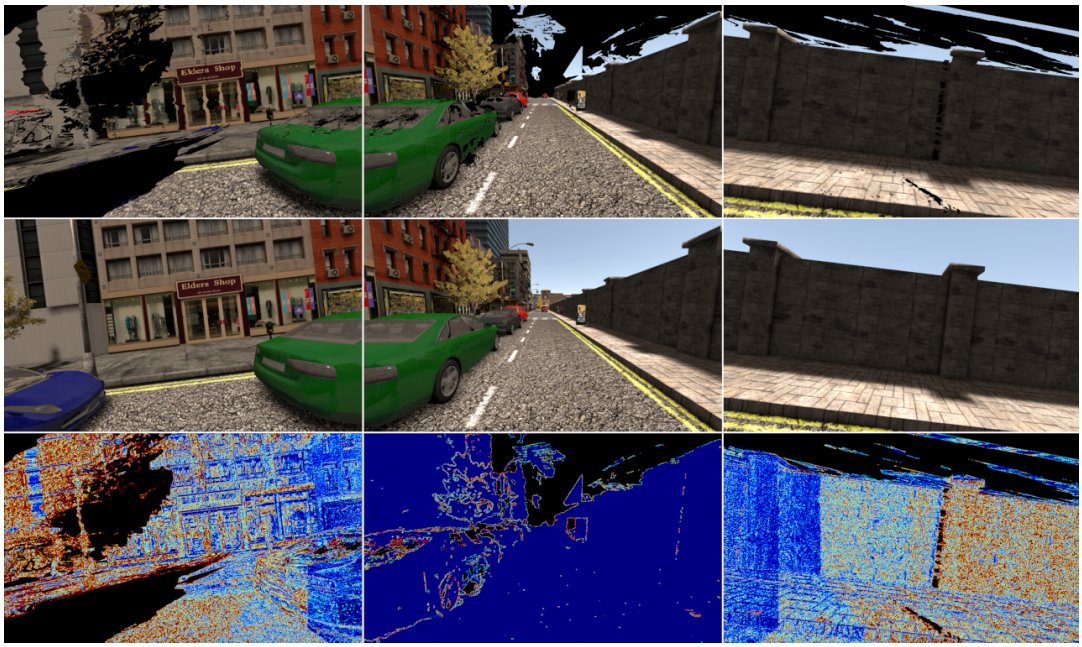


FIGURE 6.15: Illustration of view prediction error on SYNTHIA 05 with our reconstruction and ELAS as input. Left to right in row corresponds to left, front and right camera view. Top to bottom in column corresponds to the reconstruction render, ground truth image and 1-NCC error visualized (high error in red, low error in blue).

surprising because it simply is the pixel-perfect original stereo image with holes, which are rendered black throughout all evaluations.

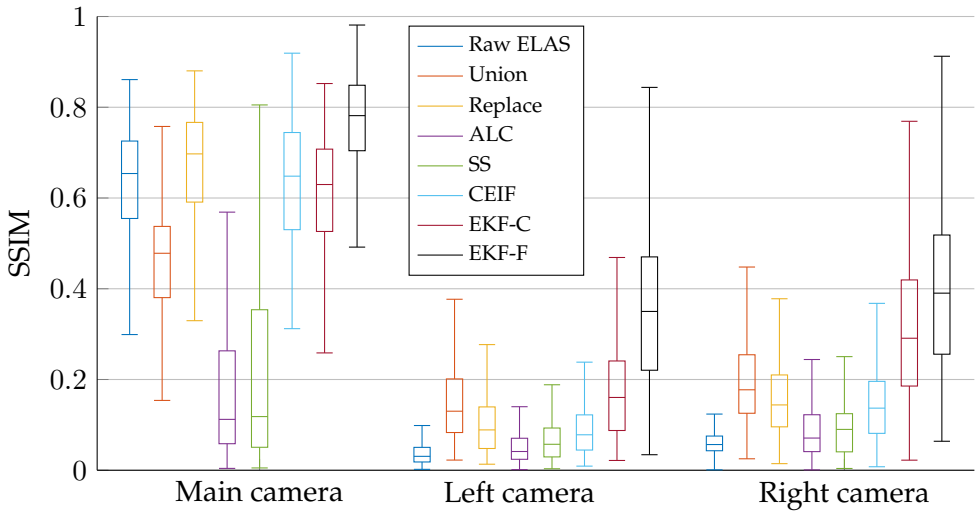


FIGURE 6.16: Boxplots of the SSIM over all SYNTHIA frames using ELAS.

In Figure 6.16, we show boxplots for SSIM on every frame of SYNTHIA using ELAS. The boxplots clearly illustrate the advantage that our mesh-based approaches have in comparison to point-based approaches for viewpoints different from the original camera viewpoint. Still, both of our **3DEKF** methods show a wide spread of actual SSIM values, which we will investigate further in Subsection 6.3.7.

Name	Front			Left			Right		
	SSIM	1-NCC	Compl.	SSIM	1-NCC	Compl.	SSIM	1-NCC	Compl.
3DEKF - M - Full	0.805	0.027	0.853	**0.379**	0.477	0.585	**0.442**	0.451	0.676
3DEKF - C - Full	0.817	**0.025**	0.862	0.378	**0.475**	0.585	0.441	**0.450**	0.671
3DEKF - Q - Full	0.757	0.064	0.844	0.145	0.618	0.287	0.175	0.589	0.358
3DEKF - Q_c - Full	**0.864**	0.020	**0.903**	0.376	0.519	**0.619**	0.439	0.491	**0.709**

TABLE 6.8: Comparison of view prediction errors and completeness of the meshing variants on SYNTHIA 05 with ELAS.

The **3DEKF-F** approach with the meshing variant M fares the best in the general view prediction error evaluation. To find out how it compares against the other meshing variants, we evaluate SYNTHIA 05 with all proposed meshing variants. This sequence was chosen because it features continuous camera motion throughout diverse urban scenes. Table 6.8 shows the evaluation results. There is no clear dominant approach. The connected quads approach Q_c achieves the highest average completeness for each view. It strictly dominates the quads approach Q in all metrics. It also performs the best overall in SSIM for the front view and is close to the best SSIM value in the side views. The mids approach M and corner C have very similar results. For the side views, both fare slightly better than the connected quads approach Q_c in the 1-NCC error, which disregards empty pixels for its assessment. C even has the lowest 1-NCC error overall. Note that we used the M variant to compare the **3DEKF-F** approach against the other reconstructors and not the Q_c variant.

The bad performance of the quad approach Q on the side views can be explained by its splat-like rendering. We orient the quads according to their observed orientation from the camera. Therefore, the front stereo camera will observe quads aligned to the camera, which are then seen differently from $90°$ to the left or right (relative to the optical axis of the main camera). The reconstruction will thin out, similar to point visualizations, as the orientation of the quad is perpendicular to the optical axis of the sidewards facing cameras. A solution could be to align the quads not according to the camera space, but along an estimated normal from the fused disparity map. Still, this would just transform Q closer to C. Note that Q_c contains parts of Q and that the improvement from Q to Q_c is due to the connecting area between the quad pixels. Also note that part of the differences for SSIM and 1-NCC can be explained by the use of bilinear filtering by M and C, while the quad approaches are effectively using a nearest neighbor filtering for the color of a pixel quad.

6.3.6 Performance

Here, we consider the performance of the evaluated reconstructors per frame. We differentiate between reconstruction and rendering time. In the quantitative evaluation, we considered only the time it takes to integrate a new depth map into the existing reconstruction and we did not take rendering speed, error metric computation or the input stereo matching speed into account. We measured the performance using a laptop with an i7-6700T CPU and an NVIDIA GTX 1080 GPU without SIMD (Single instruction, multiple data) instructions. Our render output is a single image with a resolution of $1920\,\text{px} \times 1080\,\text{px}$.

Name	Reconstruction time in ms		Render time in ms	
	KITTI VO	**SYNTHIA**	**KITTI VO**	**SYNTHIA**
Raw ELAS	9	20	13	13
Union	**7**	**15**	187	469
Replace	91	189	109	115
ALC	507	1231	100	101
SS	44	90	107	100
CEIF	241	491	105	122
3DEKF-C	218	397	8	13
3DEKF-F - M	197	397	8	13
3DEKF-F - C	193	402	8	14
3DEKF-F - C_{opt}	194	405	**5**	**11**
3DEKF-F - Q	190	394	13	22
3DEKF-F - Q_c	202	418	30	58

TABLE 6.9: Comparison of reconstruction and render time on KITTI VO and SYNTHIA (excluding stereo matching and pose computations).

Table 6.9 compares the different reconstruction approaches in terms of the time it takes to integrate a new frame into the reconstruction and the render time. Note that we use a fixed length ring buffer for 3D points for **Replace**, **ALC**, **SS** and **CEIF**, which explains the very similar render performances. **Replace** and **CEIF** remove redundancies due to the clustering and fusion, which enables it to reconstruct more of the scene with less points. This means that we can potentially reduce the ring buffer size to increase their performance further, but not for the other point-based approaches.

For the reconstruction speed, the **Union** approach fares the fastest because it just adds points to a pool of points. For **Raw Elas**, we clear all points instead of overwriting them in a large pool list, which explains the slower speed. The **3DEKF** approaches reach a frame rate of ca. 5 Hz on KITTI and ca. 2.5 Hz on SYNTHIA, which corresponds to half of the real time speed of the respective dataset. **ALC** is the slowest because it uses expensive geometrical and photometrical checks between frames.

For the rendering speed, our mesh-based approach clearly outperforms other approaches when not using the more expensive Q or Q_c variant. **Union** takes a long time to render because of the unoptimized point rendering. Note that our optimized mesh corners approach C_{opt} needs a little more overhead in the reconstruction time compared to C to limit the number of elements during draw calls. It represents our fastest render time with 5 ms (ca. 200 Hz) on KITIT VO and 11 ms (ca. 91 Hz) on SYNTHIA, which is indeed in the range of the display refresh rate of modern HMDs. Thus, we have accomplished one of our main goals for immersive stereo vision. Furthermore, **Raw Elas** again does not represent a 3D reconstruction over multiple frames, but is trivially fast because we only have to render points in the order of a single image. Also note that for the determination of the render times, we had VSync enabled and used a 250 Hz monitor for this test, which means that our refresh rate

was capped at 250 Hz and could potentially be even faster without VSync, *i.e.* without waiting for the new frame on the monitor.

FIGURE 6.17: Percentage of reconstruction time spent on one iteration using our full 3DEKF M approach on SYNTHIA with ELAS as input.

For a more detailed breakdown of the reconstruction time, we show the average percentage of time spent on each part of the computation of one frame in Figure 6.17. We break down the hierarchy of the biggest block even further. The main part of the iteration can be done in parallel for each pixel / bin, which is why we use OpenMP [Ope19] to start a parallel block (big green block). In this block (blue elements), we first have to integrate the warped active region and the new stereo measurements from before. Note that their time scales approximately with the number of pixels, *i.e.* the foreground is generally denser than the new stereo measurement for ELAS and the background has the least pixels. To add, an extra middleground layer would also be mostly empty in most situations. A huge part of the run-time of the blue blocks is the clustering stage, which can be broken up again in sub-tasks (orange elements). Here, we associate measurements unsupervised and fuse them, which is again broken down in four stages in yellow.

Optimization potential might still exist for the clustering because we could constrain the bins to only a maximum of three points (new stereo measurement, FG and

BG), which might speed up adding new clusters with a better data structure. However, a full GPU integration tightly coupled to OpenGL would be possible because of the parallel nature of the image and render operations. Also, it would enable us to skip the download of the rendered offscreen images from the GPU, aswell as the preparation and upload of new images to the GPU.

6.3.7 Ablation Studies

Since we are close to real-time reconstruction performance on KITTI, it is interesting to see how the reconstruction quality degrades if we skip frames for performance reasons. Additionally, we may be able to use a coarser mesh resolution since we are sampling a comparatively high-resolution texture in the fragment shader. We also test how a distance-based dynamic frame skip fares compared to our standard method of adding every frames to a limited size ring buffer.

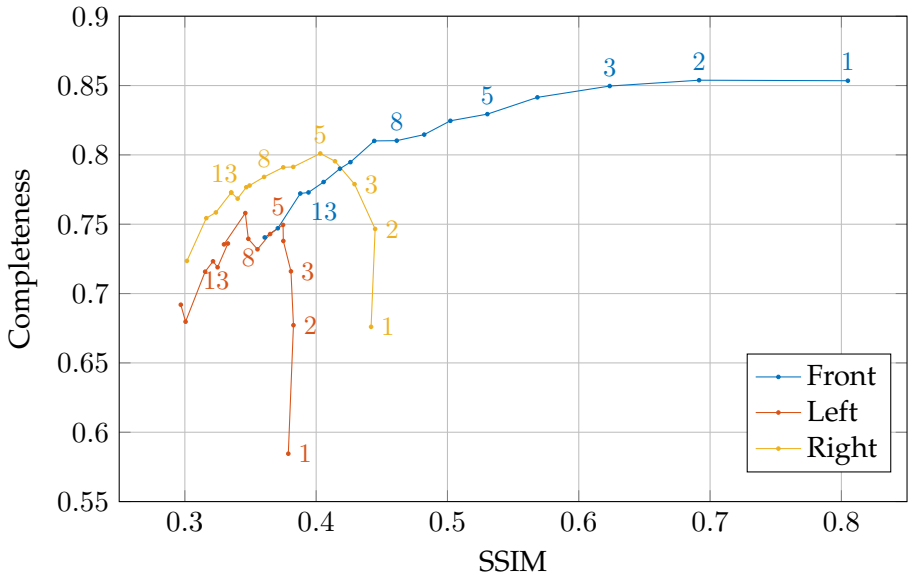


FIGURE 6.18: SSIM and completeness for different frame step sizes on SYNTHIA 05 using our full M approach.

In Figure 6.18, we display the SSIM and completeness of our full M mesh variant approach with different frame step sizes (1 to 16), *i.e.* the new frame index is computed by adding the associated step size. The view prediction error evaluation still performed for every frame. Within the first few step sizes, the front view degrades fast in SSIM, but only slowly in completeness. The left and right view both behave differently: The SSIM degradation is not as fast as the degradation of the front view, and the completeness surprisingly increases with larger frame jumps. This can be explained by longer stops or slowdowns in the sequence. The tube mesh consists of a fixed ring buffer size, *i.e.* it contains a fixed number of frames, which means that the covered area of the tube mesh will shrink when adding new frames while stopped. One possible improvement could be to use the pose movement to trigger adding

a new tube mesh frame only when reaching a certain threshold for pose changes, which we will evaluate later in this section.

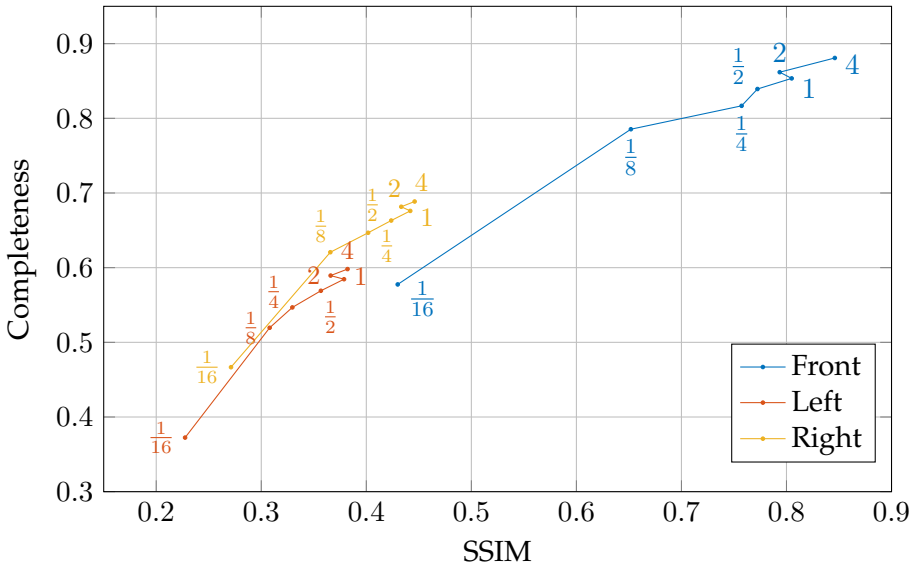


FIGURE 6.19: SSIM and completeness for different mesh resolutions on SYNTHIA 05 using our M-Full approach and ELAS as input.

Figure 6.19 shows the view prediction error of our full M mesh variant approach with different mesh resolutions, *i.e.* using $\frac{1}{16}$ to 4 ($2^i, i = -4, \ldots, 2$) times the amount of given pixels. For example, a mesh resolution 4 times as high would subdivide each pixel into four smaller squares. Generally, a higher resolution results in a better completeness. This can be explained by the fact that triangles at depth edges can now be cut out more precisely and the omitted triangles in the geometry shader are smaller. Here, using only $25\,\%$ of the original pixels for the mesh offers a potential great trade-off between computation and completeness/SSIM. The completeness loss of the lowest resolution version can be explained by our triangle thresholding: Depth differences are amplified because of missing intermediary points. Even triangles from smooth surfaces such as roads are less likely to pass the valid triangle test because of their orientation to camera space.

We implemented a dynamic frame skip by accumulating absolute translational differences to the last reconstructed camera pose until its magnitude is larger than a certain threshold in meters. Note that rotation changes of the vehicle will also result in translational changes, *i.e.* yaw changes, which is why we only consider translation. We test minimum distances from $0.015\,625\,\mathrm{m}$ to $16\,\mathrm{m}$ ($2^i, i = -6, \ldots, 6$) as shown in Figure 6.20 and also plot the reference value without skipping frames. The resulting count of frames skipped on SYNTHIA 05 can be seen in Table 6.10. The sequence consists of 786 frames in total.

First of all, the front view slowly degrades in Figure 6.20 with bigger frame skips as more of the scene is missing, which could be filled from previous frames, or because the scene texture resolution is worse because of the lack of close observations. The left and right view behave very similarly, but the right view fares generally better

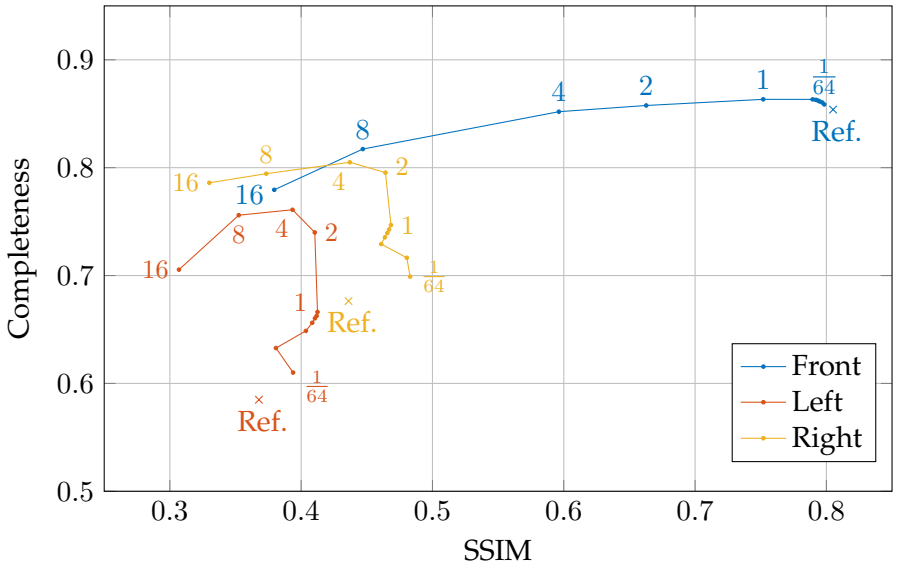


FIGURE 6.20: SSIM and completeness for different minimum distances for our dynamical frame skip on SYNTHIA 05 using our M-Full approach and ELAS as input. Reference with no skips marked as Ref.

Min. distance in m	**Skips**	**Ratio in** %
0.015 625	56	7.1
0.031 25	72	9.2
0.0625	83	10.6
0.125	90	11.5
0.25	98	12.5
0.5	104	13.2
1	120	15.3
2	430	54.7
4	555	70.6
8	668	85.0
16	727	92.5
Reference (no skips)	0	0

TABLE 6.10: Overview of number of skipped frames using our proposed dynamic frame skip method on SYNTHIA 05, using ELAS and 3DEKF-F with the M variant.

in this dataset. This could be because of the relatively simple geometry of a wall in the scene. Moderate dynamical frame skips for the side views improve both the SSIM and completeness compared to the reference reconstruction. This is because the sequence features stops at which the tube mesh would collapse to a single tube mesh element. Here, we could prevent this behavior by using a dynamical frame skip of up to 2 m without sacrificing SSIM significantly to reconstruct the scene with a higher completeness. Of course, the mesh texture resolution may not be on par compared to the reference, but a minimum distance of 2 m would allow us to skip 54.7 % of the frames. Unfortunately, this performance gain may not help us to achieve real-time reconstruction speed consistently because the frame skips are not equally distributed, but mostly at stops or lower speeds.

6.4 Qualitative Evaluation

A major assumption we held so far is that if we improve the quantitative reconstruction quality, *e.g.* the geometric quality or the view prediction error, we also improve the relevant visual aspects, which are very important for immersive use-cases. This section examines this assumption. For that, we showcase and compare reconstructions qualitatively on different datasets and also discuss observed limitations. For illustration purposes, we show more tube elements than necessary for our target application in this showcase, *i.e.* geometry that we have passed with the vehicle can be removed much sooner than depicted when viewed from inside a vehicle.

6.4.1 Showcase on High Stereo Baselines

First, we showcase different 3D reconstructions of scenes on KITTI VO and SYNTHIA, which we already introduced for the quantitative evaluation. These datasets feature stereo baselines that may be too large for conventional car designs, but nonetheless may be practical for vehicles such as trucks.

KITTI 2012 (Visual Odometry)

The KITTI VO dataset features a large stereo baseline of ca. $0.54\,\mathrm{m}$. For the showcase, the semantic segmentation of DeepLabv3 [Che+17] is used for the removal of detected sky pixels. We show a variety of different KITTI VO scenes using our 3DEKF-F M approach and GA-Net as input in Figure 6.21. The scenes contain urban and sub-urban scenery with some vegetation and traffic signs in them. GA-Net handles reflections at specular surfaces like vehicle windows very well. Traffic signs are not duplicated, which was part of the main motivation for our fusion approach. The tube mesh transitions are not that noticeable in the scenes and there are no apparent gaps. The reconstruction of the road appears to be very consistent. The vegetation makes a good visual impression as well. Still, some artifacts remain, *e.g.* in the image on the mid left in Figure 6.21: Here, very elongated triangles with a uniform color are drawn next to the traffic sign, which can be caused by previous bad stereo matches. Also, some of the sky is still reconstructed even when using a semantic segmentation. We think, to the best of our knowledge, that this is the best looking 3D reconstruction of the KITTI dataset up until today.

FIGURE 6.21: Showcase of different scenes on KITTI VO using our 3DEKF-F M approach with GA-Net as input.

SYNTHIA

SYNTHIA has a huge stereo baseline of ca. $0.8\,\mathrm{m}$. Nonetheless, we have to rely on ELAS as our stereo input for SYNTHIA, which lacks completeness in comparison to GA-Net. Figure 6.22 shows the reconstruction of three different scenes using our 3DEKF-F M approach and ELAS as input.

The top image is a reconstruction of a left-turn at an intersection from a very high perspective. The road and house facades are reconstructed quite consistently. Stereo shadows can be seen due to the very different virtual camera perspectives compared to the poses of the source stereo camera. From this top-perspective, we can see that the bottom-right part of the image has a lower resolution overall. This is because this part of the scene is only observed from afar.

The middle image shows a reconstruction of a small tunnel from a higher perspective. The active region mesh, *i.e.* the current stereo camera, is already at the end of the tunnel. The tunnel itself looks very consistent. Some holes appear due to homogeneous textures, *e.g.* the windows on the facades and the right house wall.

In the bottom image, the virtual camera is now placed much closer to the road. The current stereo camera is directly behind the red vehicle. The road itself again looks very consistent. The sky features a lot of artifacts and the far away hill range along the highway looks consistent. The lamp post and traffic sign appear to be

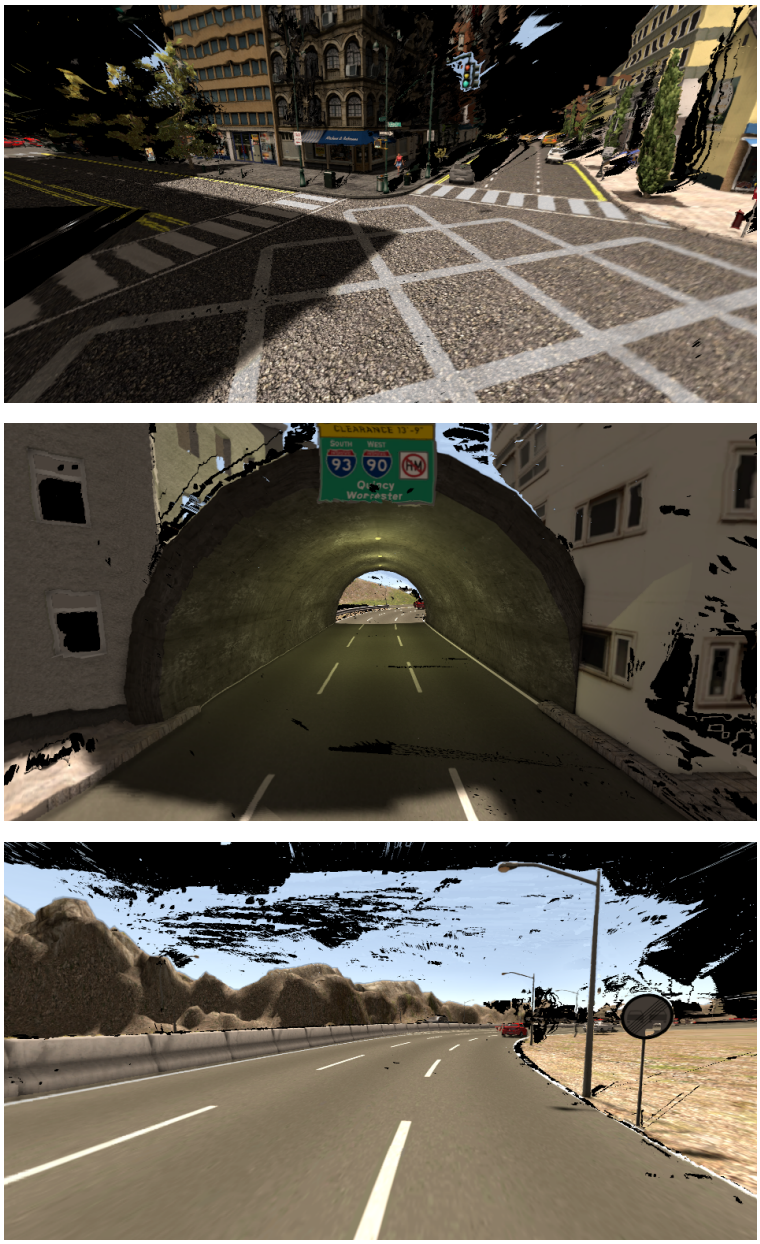


FIGURE 6.22: Showcase of different scenes on SYNTHIA using our 3DEKF-F M approach with ELAS as input.

mostly reconstructed well, only featuring a small hole in the front sign. The first lamp post is not perfectly cut out and has some fringe-like sky artifacts directly surrounding it, which are not that visible from this perspective.

No tube mesh transitions can be seen throughout all images of Figure 6.22. SYNTHIA does not simulate high dynamic range effects, which we have encountered in KITTI (auto-exposure). It also features no vignetting, which is a fall off in brightness towards the image edges. This allows us to have almost invisible tube mesh transitions.

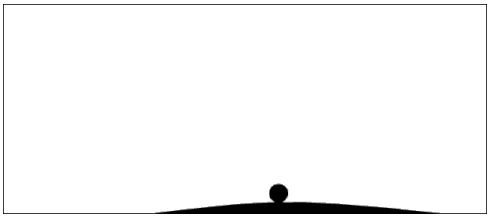
6.4.2 Showcase on Low Stereo Baselines

Here, we evaluate our 3D reconstruction approach on our own recorded data with much smaller stereo baselines to test how it generalizes in an actual real-world context. The first scenario features an automotive stereo camera, which, unlike KITTI, cannot be mounted on top of the vehicle. The second scenario is using a compact stereo camera attached to a bike helmet to capture stereo video while cycling.

Automotive Stereo Camera

There are various considerations to take into account for the integration of a front-facing stereo camera in a vehicle. For instance, the baseline is limited to not obstruct the view of the driver and to still be in the area covered by the windshield wipers. Here, we use a stereo camera that is integrated behind the windshield of a test vehicle. The image sequence consists of images captured at ca. $16\,\mathrm{Hz}$ with a resolution of $1920\,\mathrm{px} \times 832\,\mathrm{px}$. The images feature a small part of the hood, which can be statically removed via a mask. An example of the masking is shown in Figure 6.23. Here, blackening the hood pixels in the left image before stereo matching achieved an accurate exclusion of the hood in the depth image. In contrast, masking out pixels after the stereo computation, *i.e.* on the depth image, can result in artifacts caused by matched hood pixels that extend further than the actual hood when using ELAS.

(a) Left RGB image.



(b) Hood mask for left images.

(c) Masked left RGB image.

FIGURE 6.23: Masking of left RGB images using the automotive stereo camera.

The used focal length is approximately $1450\,\mathrm{px}$ and the stereo baseline ca. $0.21\,\mathrm{m}$, which is roughly only a third of the baseline in KITTI VO, but still quite large for an automotive stereo camera. The original color images had a higher bit depth than 8 bit. The images were resampled to 8 bit to ensure compatibility with standard stereo algorithms. For the poses, we tested multiple open-source stereo VO algorithms, which are described in Planz [Pla19], on this sequence. ORB-SLAM2 [MAT17] with default

parameters in SLAM mode achieved the best results in terms of pose consistency, which is why we employ it here. Note that we are using SLAM and computing the poses offline. In practice, we could rely on more vehicle sensor data for an accurate localization.

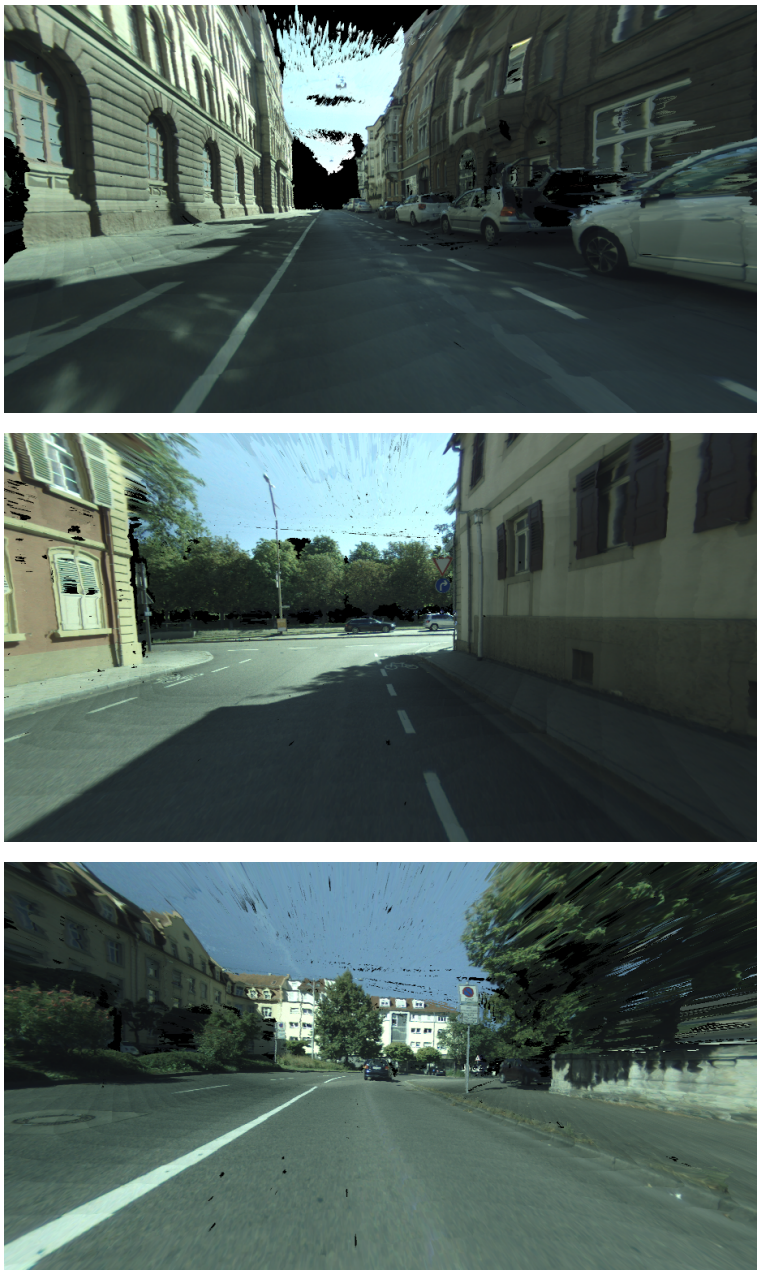


FIGURE 6.24: Showcase of different scenes with the automotive stereo camera using our 3DEKF-F M approach with ELAS and ORB-SLAM2 as input.

We showcase three results of the automotive stereo camera in Figure 6.24. All images were taken near street-level. The road surface looks very consistent. Tube mesh transitions are quite noticeable since we only use VO for the pose estimates. We can also observe image vignetting in the corner of the input images. In the top image, ELAS struggles to consistently estimate the opened trunk of the vehicle. The

house facades appear to be well-reconstructed, but the top parts of the facades appear to be skewed and in lower resolution in both the top and mid image. This can be explained by the fact that these parts of the facade were only observed from very far away compared to the street-level facade parts, which can be observed up close.

Helmet Stereo Camera

For an additional, different scenario next to the automotive context, we attach a Stereolabs ZED Mini [Ste19] to a regular bike helmet and connect it to a laptop in a backpack with an nVidia GTX1080 GPU. In Figure 6.25, we show the stereo camera, which is attached to the helmet. The ZED Mini is a compact color stereo camera that is designed for the capture of Mixed Reality content via stereo pass-through (when attached to a VR HMD). It features a stereo baseline of $0.063\,\mathrm{m}$, which is only an eighth or ninth of the KITTI VO baseline, but corresponds to the human interpupillary distance. Also, the camera uses a rolling shutter. Still, it allows for higher frame rates and higher resolutions than the stereo camera in KITTI VO. The ZED Mini also has a gyroscope and accelerometer, which enables visual-inertial stereo SLAM methods. The accompanying SDK from Stereolabs provides motion and depth estimates. Here, we will use the pose estimate of the SDK since it incorporates the IMU measurements. Still, we use ELAS as our stereo matching algorithm because it handles the road surface better than the SDK depth estimation in our tests.

FIGURE 6.25: ZED Mini stereo camera attached to a bike helmet.

For the recording, we used the helmet setup on a bike and drove straight along a bike path on a sunny winter day. For the camera settings, we used a constant exposure,

a frame rate of $60\,\mathrm{Hz}$, a resolution of $1280\,\mathrm{px} \times 720\,\mathrm{px}$ (corresponds approximately to a focal length $f = 700\,\mathrm{px}$) and recorded an SVO file (ZED SDK) using the native "ZED Explorer" application of the SDK with H.265 encoding enabled. The SVO file contains all compressed stereo frames and inertial measurements, which allows us to replay the stereo video offline.

FIGURE 6.26: Showcase of different scenes with the helmet camera using our 3DEKF-F M approach with ELAS and the Stereolabs SDK poses as input.

For the reconstruction, we processed every stereo frame but only used a quarter of the resolution for the tube mesh to visualize more of the scene due to the high frame rate. The results can be seen in Figure 6.26. Note that we used a depth cut-off at $40\,\mathrm{m}$, which means we still show far away objects despite the relatively tiny stereo baseline of the camera. The top images show the same scene from different perspectives. The road surfaces look well reconstructed. The sides of the images reveal more distorted scene content and visible tube mesh transitions. The sign in the bottom-left image is well cut out, but has a distorted base. The sky is again only reconstructed because of erroneous depth estimates of ELAS. These examples prove that it is possible to also use our algorithm for other front-mounted scenarios such as cycling.

6.4.3 Comparisons

In this section, we compare the reconstruction results of different reconstruction approaches and different parameters seen from the same perspective. First of all, we emphasize the missing pixels in Figure 6.27 to compare the completeness measure directly in images. The magenta blob in the middle is due to our depth limit of $50\,\mathrm{m}$. Note that the input **Raw ELAS** has missing pixels in the bottom right corner of that frame. This translates to sampling holes visible in all point-based reconstructions and also **3DEKF-F** Q since it is essentially a splatting technique. **ALC** shows the lowest

completeness with a strong bias for near depths and **Union** covers the most pixels with lots of spurious 3D points. Comparing the **3DEKF-F** approaches, **3DEKF-F** Q_c is able to not show streaky artifacts and also reconstructs more of the house facades.

(a) Raw ELAS. (b) Union.

(c) Replace. (d) ALC.

(e) SS. (f) CEIF.

(g) 3DEKF-F M. (h) 3DEKF-F C.

(i) 3DEKF-F Q. (j) 3DEKF-F Q_c.

FIGURE 6.27: Comparison of completeness of different reconstructors using ELAS on KITTI VO 01. Empty pixels are highlighted in magenta.

These streaks can be explained by the dismissal of steep triangles in the geometry shader. For **3DEKF-F** Q_c, the pixel connecting meshes are omitted in these areas, which allows the actual pixel meshes to be rendered normally. This is not the case for the M and C variant, where we have to remove part of the actual pixel. The C variant features slightly larger streaks than the M variant. This may be caused by the bilinear filtering, in which a depth edge could influence more than a single pixel.

In Figure 6.28, a traffic sign, which is a highly salient and recognizable object in the scene, is reconstructed with all the other point approaches and our full mesh approaches (**3DEKF-F**). We see the traffic sign from a higher perspective than the original source camera and also look at it slightly from the left. The points are drawn

(a) Union. (b) Replace. (c) ALC.

(d) SS. (e) CEIF. (f) 3DEKF-F M.

(g) 3DEKF-F C. (h) 3DEKF-F Q. (i) 3DEKF-F Q_c.

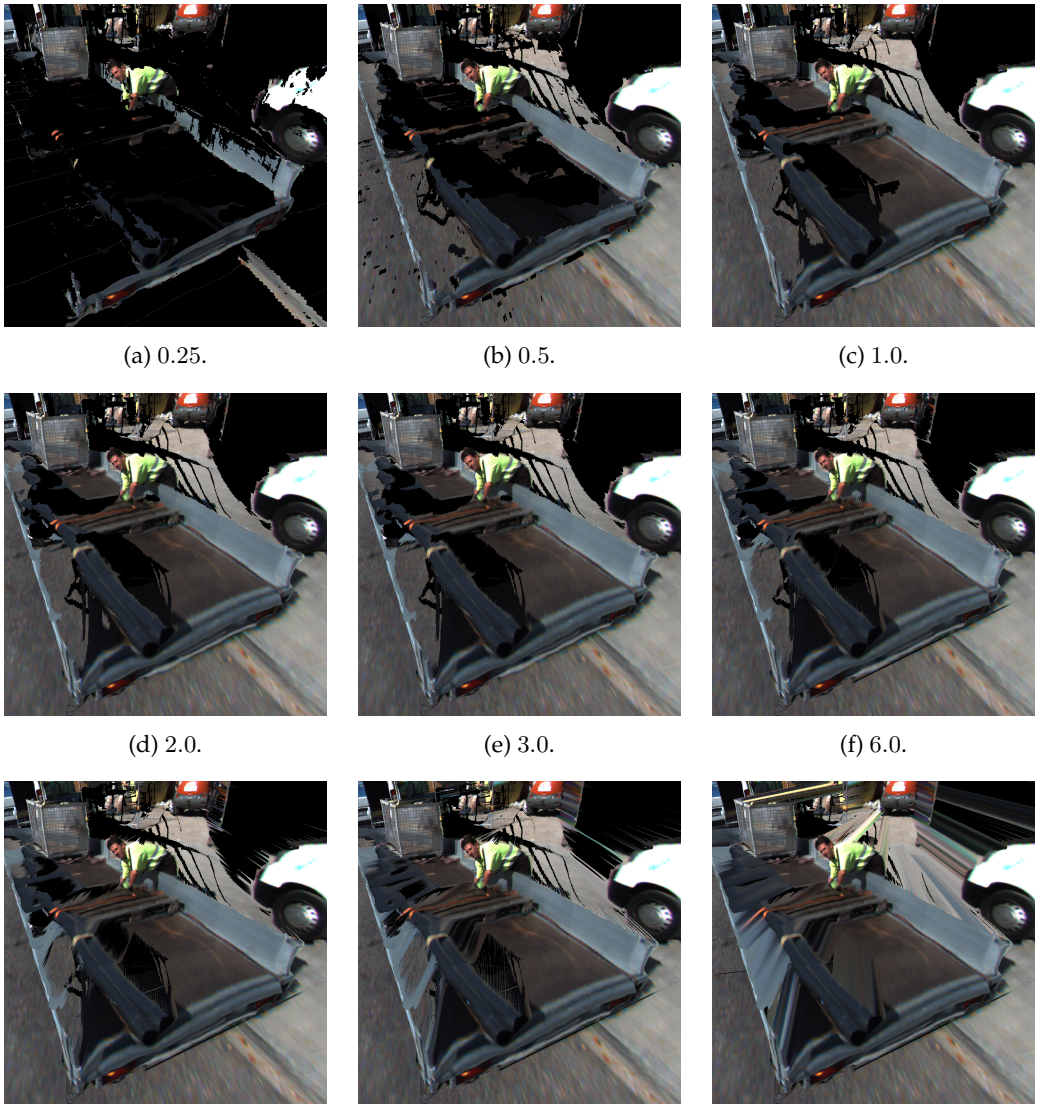
FIGURE 6.28: Comparison of the different reconstructions of a traffic sign on KITTI VO 06 with GA-Net, using a *glPointSize* of 3 for the point clouds.

bigger than the default size so that the traffic sign occludes the background scenery. As expected, the **Union** approach shows lots of noise over multiple frames and **SS** follows a similar pattern. **3DEKF-F** Q should be similar to point-based approaches, but shows even more holes due to the quads being aligned to the image space of the source camera and not the virtual camera.

Generally, the mesh approaches make a better visual impression. The traffic sign is not perfectly cut out, which is due to the stereo matching properties of GA-Net. Here, a foreground fattening is induced. A depth edge refinement could improve the appearance, but preliminary tests had adverse effects on other parts of the scene. **3DEKF-F** M and C look very similar, while Q_c has differently formed triangulation artifacts and a different texture sampling (*i.e.* nearest neighbor). Also note that **ALC** does not reconstruct the background during this sequence and that the frame

transition is generally directly visible on the traffic sign, especially in **ALC** on the top part of the triangle.

For our image triangulation visualization, we have to remove triangles at strong depth edges to achieve an actual 3D representation using 2.5D depth image triangulations. Here, we vary the threshold for the maximum allowed disparity difference in a triangle, which will still be rendered. Generally, we use a threshold of 3.0 times the disparity error e_d, which corresponds to the clustering threshold T_c. This means that we only show triangles if we would cluster their vertices in disparity, *i.e.* if we would regard them as belonging to the same depth layer.



(a) 0.25. (b) 0.5. (c) 1.0.

(d) 2.0. (e) 3.0. (f) 6.0.

(g) 12.0. (h) 24.0 (i) No check.

FIGURE 6.29: Effects of different triangle consistency thresholds on KITTI VO 06 using GA-Net and our full 3DEKF M approach.

Figure 6.29 shows the effects of various thresholds using a virtual camera, which is located higher up and pitched down compared to the original source camera. This virtual camera looks partly into unobserved areas. Thresholds 0.25 and 0.5 have large holes in the reconstruction compared to the other images. 0.25 mostly

reconstructs objects with very similar depths. Starting from 1.0, the reconstruction closes additional gaps, but at the expense of introducing spiky artifacts. These are caused by relatively large depth gaps compared to the closely sampled pixels. Having no threshold at all will result in big triangles that connect the foreground to the background, which may fill holes but also block the view from different perspectives like a 2.5D representation. Foreground and background objects are directly connected as a result in the reconstruction without the check.

6.4.4 Limitations

This section deals with various observed limitations of our approaches and 3D reconstruction itself in our scenario. For that, we show samples of reconstructions with our **3DEKF-F** M approach and discuss the cause of each visible artifact.

Stereo Failures and Tube Meshing Artifacts

Here, we highlight some of the failure cases of our tube meshing caused by stereo matching errors. We cannot recover from consistent observation errors and have to rely on well-natured stereo matching input to achieve an immersive reconstruction. One example for this is shown in Figure 6.30. Here, we have kept the tube mesh very long and the current camera position has already traveled past the virtual camera. Thus, we visualize more scenery than needed in regards to our target application. The thin mesh structure of the construction site fence is notoriously difficult to estimate for stereo matching algorithms because the majority of the surrounding pixels have a very different depth and its repetitive nature leads to stereo matching ambiguities. However, the poles of the construction site fence are still correctly estimated. Here, the actual fence texture is wrongly mapped to the white facade in the background.

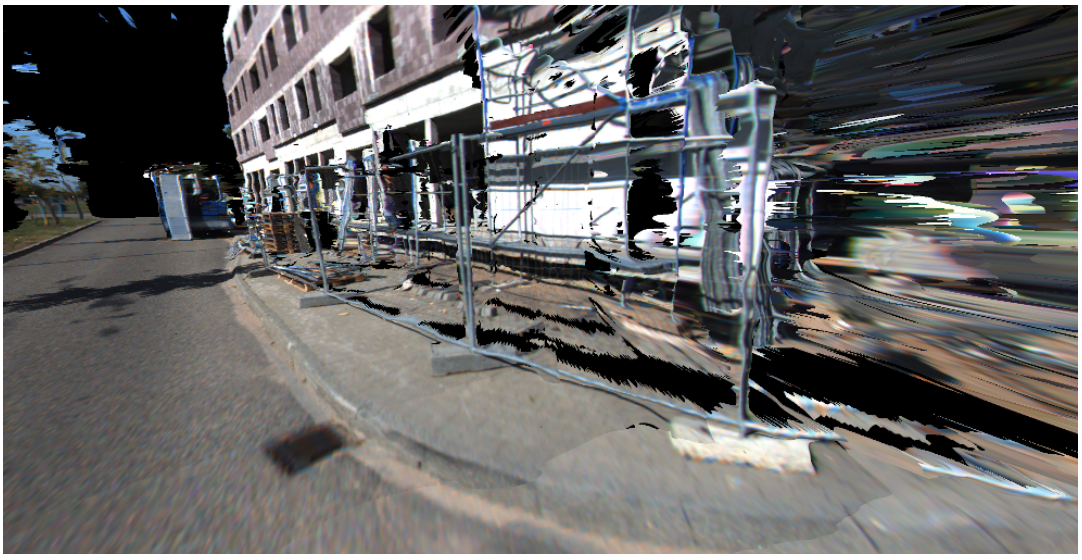
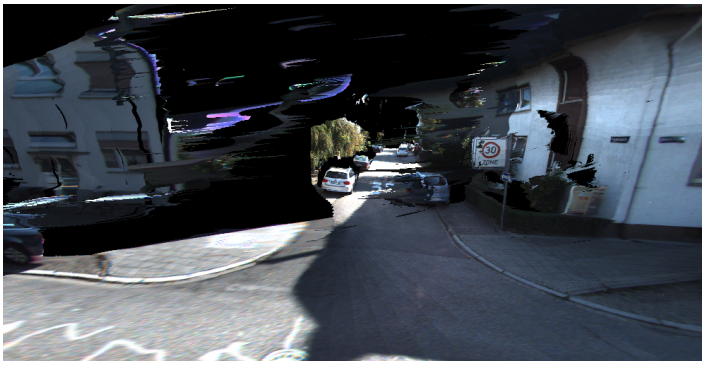


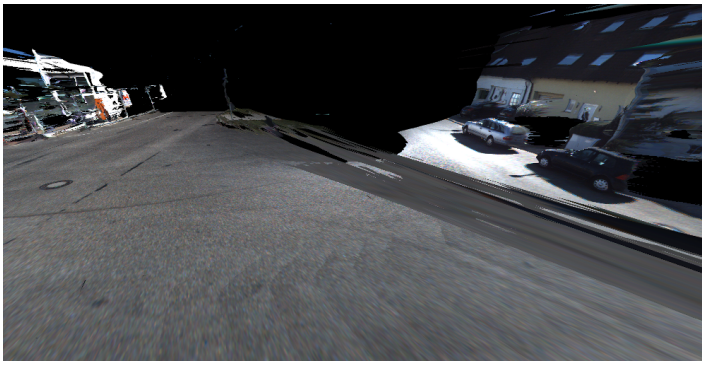
FIGURE 6.30: Challenging scenery for stereo matching, using GA-NET and 3DEKF-F on KITTI VO 04.

Figure 6.31 shows examples how stereo outliers from far away are now occluding parts of the correct scene geometry during vehicle turns. These artifacts are homogeneously textured spikes that still pass the triangle test in the geometry shader stage because they consist of multiple similar outlier pixels. Generally, these spikes can be reduced by shortening the size of the tube mesh, *i.e.* making it consist of fewer frames of the past.



(a) 90° turn.

(b) Wide 180° turn.



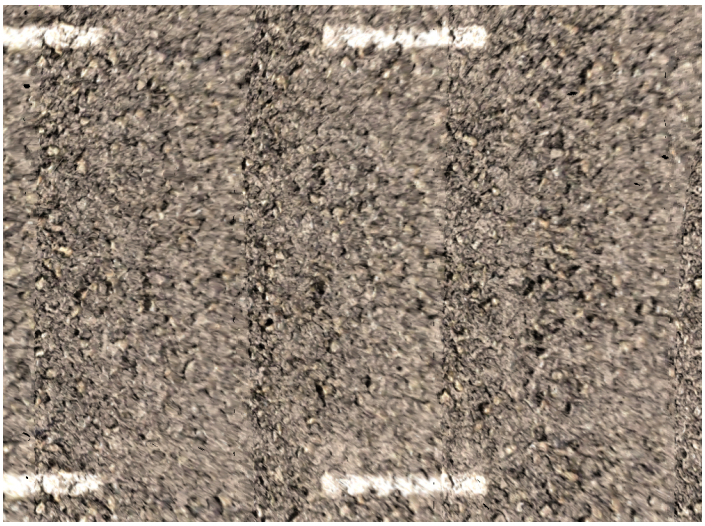
(c) Sharp 180° turn.

FIGURE 6.31: Examples of observed stereo artifacts causing spurious meshes during turns, using GA-NET and 3DEKF-F on KITTI VO.

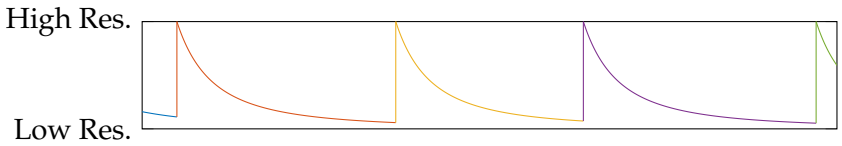
Resolution

Automotive stereo cameras are usually aligned so that their optical axis is parallel to the planar road surface. This is because pitching the camera up would show

more of the unnecessary sky and pitching it down would show more of the hood. The reconstructed resolution of close scenery is much higher compared to far away scenery. This is because the ray volume of a pixel grows with increasing distance and this volume has to include more scenery physically. Furthermore, for an even road, the angle between a camera view ray and a normal of a road surface scene point can be close to $90°$ depending on the distance, which further worsens the pixel density for the reconstruction due to perspective distortion. When rendering an image of a road surface with to our depth image triangulation, the lower pixel rows will have a higher density along the vertical direction when backprojected, while upper rows are more and more elongated along this direction. This is analogue to a beamer at the camera position that projects a grid pattern in image space on the road surface: The individual grid rectangles will appear as trapezoids with increasing area depending on the distance.



(a) Top-down view on reconstructed road surface.



(b) Approximate relative resolution in horizontal direction.

FIGURE 6.32: Resolution jumps and falloffs due to perspective projection in individual tube mesh elements.

The top image in Figure 6.32 shows a top-down view of a road surface reconstruction using the tube mesh of our **3DEKF-F** M approach. Here, the camera moved from left to right while looking in the direction to the right. The road surface shows four sharp tube mesh transitions along the vertical image direction. The pixels to the left of a transition appear to be stretched. In the bottom figure, we illustrated how the resolution changes qualitatively along the horizontal direction. Each new tube mesh element is colored differently and first appears with a high resolution peak with a subsequent falloff. It resembles a sawtooth wave with a negative ramp, but with a

falloff that is quadratic instead of linear because of the perspective projection at this inclination angle.

These resolution jumps may be noticeable to a close observer and may impact the feeling of immersion. A straightforward solution would be to artificially limit the resolution with a filter to the lowest resolution, which may itself have negative consequences for the immersion because of the loss of details.

Missing Near Depths

Subsection 4.6.2 dealt with the theoretical consequences of the non-linear disparity sampling using our EIF. In essence, our confidence measure builds confidence up to a point until it decays strongly because we trust new closer stereo observations more than the fused observations from far away up to this point. Using a static confidence-based threshold, we may omit very close and accurate depths.

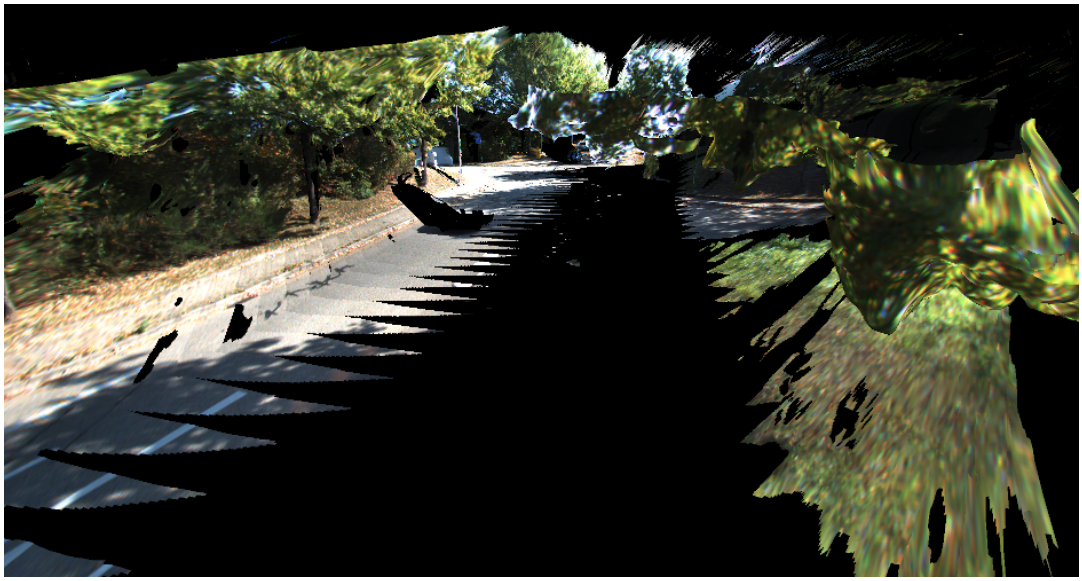


FIGURE 6.33: Reconstruction over multiple frames on KITTI VO with missing near depths of 3DEKF-C M using GA-Net.

Figure 6.33 illustrates a particular bad situation with **3DEKF-C**, which is our confidence-based thresholded method. Clearly, parts of the road are missing in the tube mesh (passive region), but dynamic objects like the passing car are removed because these points are inconfident. This effect can be alleviated to some extent by decreasing the pointing errors σ_u and σ_v or decreasing the translational pose error, giving us a more complete reconstruction. This will obviously result in improvements in depth errors (very near geometry), but at the cost of disparity errors. Note that this does not directly affect the stereo error benchmark results because the active region mesh is still quite dense.

We already discussed some alternatives to a static confidence threshold. Again, in our experiments, none of these ad hoc measures improved the disparity error or normalized depth error of **3DEKF-C**. Again, we are in a dilemma: We are trading

off depth over disparity and it is inconclusive which approach is generally better. Previously, we have associated one disparity error metric with the ground truth disparity value itself in Figure 6.12 using ELAS on KITTI VO. The figure suggests that the depicted error increases with higher disparities on KITTI VO and that **Raw ELAS** outperforms our **3DEKF-C** over $70\,\mathrm{px}$ (below ca. $5.4\,\mathrm{m}$), which explains that trade-off. Small gains in that error metric can be theoretically made if we could just replace points of **3DEKF-C** with **Raw ELAS** during evaluation, but since we are using median values, we cannot reason about the prevalence of outliers. The same figure produced with GA-Net instead of ELAS has very similar results.

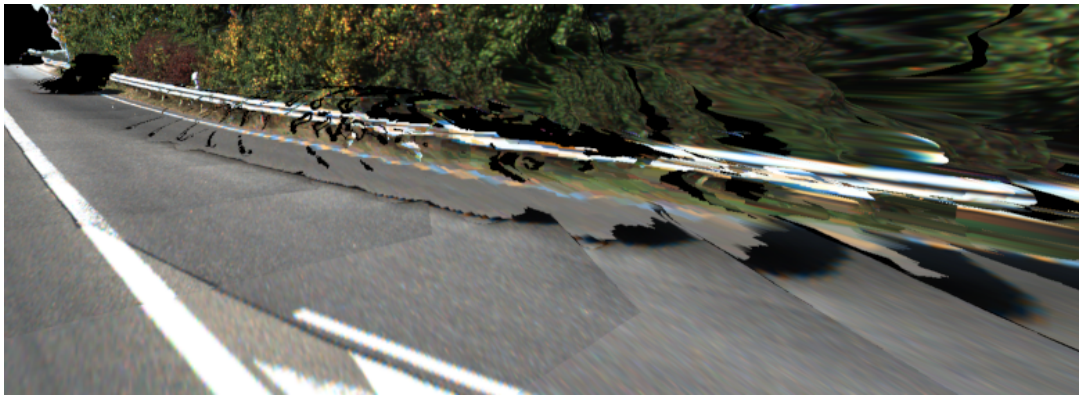
We assume here that the median of the disparity median errors increases due to inaccuracies in the calibrations in KITTI VO, which makes it hard to make clear-cut statements regarding the aforementioned trade-off because the disparity error does not remain static. It actually increases for closer geometry, *i.e.* higher disparities. Similarly, but not quite as dramatic, the same figure using ELAS over all frames of SYNTHIA produces a small linear increase of the median of disparity median errors with increasing ground truth disparity for all reconstructors. This again violates our assumption of a static disparity error, which we assume to be independent of the disparity itself in our probabilistic filters. Further research is needed why exactly this is the case and how we can potentially cope with it in our probabilistic filter models. Generally, the influence of the true disparity on the disparity error itself might exist because a true high disparity means more potential matching candidates, *e.g.* a greater search window. With a higher disparity, the likelihood increases that a similar 3D point is preferred to the actual matching 3D point when looking for stereo correspondences.

Dynamic Objects

The top image Figure 6.34 shows a reconstruction of a scene with a dynamic object, which was located at the transition between the active and passive region over multiple frames. Here, this means that the back of the vehicle will be part of the passive region in the next tube mesh element. This part of the vehicle will not be seen again, which is why we cannot use a freespace check to remove the resulting ghosting artifacts to clean up the reconstruction. As an alternative, we can use semantic image segmentation to remove image regions classified as dynamic objects as proposed by Bârsan [Bâr+18].

The bottom image in Figure 6.34 shows the resulting reconstruction with removed dynamic objects. Unfortunately, the simple removal of dynamic objects is not a perfect solution: The shadows of the vehicle are not classified as dynamic objects and still appear as artifacts. This means that for our purposes we would need to extend the standard labels to include shadows or introduce a new label, which can be laborious. We would also have to retrain the neural network. Expanding further, the removal of a 3D object from the scene may also create more subtle artifacts that are harder

(a) Original reconstruction with ghosting artifacts in the tube mesh.



(b) Removal of dynamic objects.

FIGURE 6.34: Ghosting artifacts on KITTI VO 01 using GA-Net, our full 3DEKF M approach and the semantic segmentation of DeepLabv3 [Che+17] for the removal of dynamic objects.

to remove, *e.g.* lighting effects such as ambient occlusion or the effects of reflective material on the dynamic objects.

Furthermore, the removal of the object reveals the occluded scene as shown in Figure 6.34, which is both geometrically less accurate and has a lower texture resolution since it was only observed from a greater distance due to being occluded by the dynamic object.

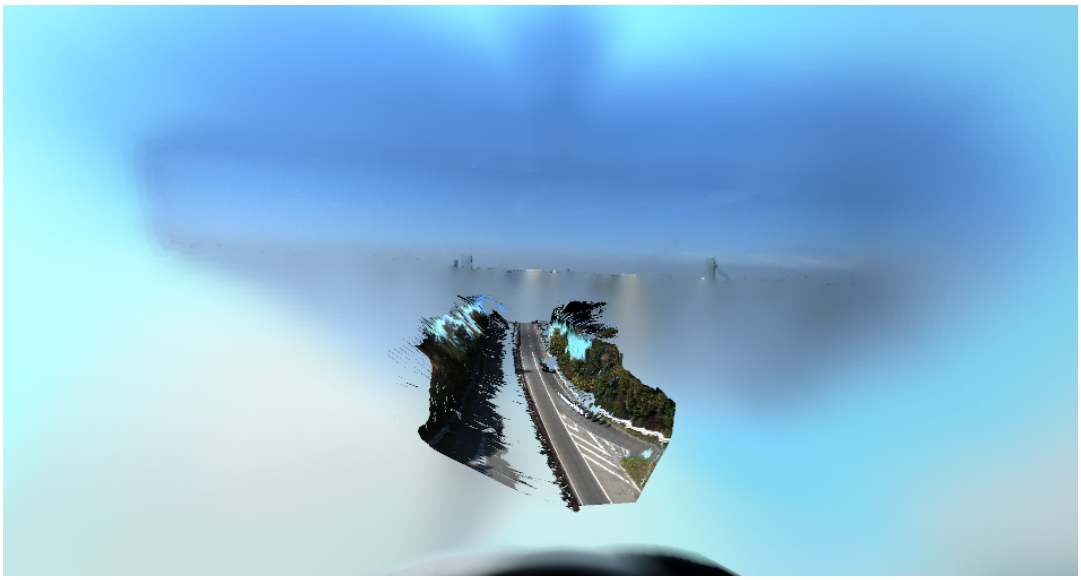
Virtual Sky Artifacts

The KITTI VO image sequences prove to be prone for color changes during the skybox texture diffusion for the virtual sky. This is caused by the automatic exposure setting of the stereo camera. This setting can result in very inconsistent sky color changes within a few seconds, *e.g.* in just 2 s as depicted in Figure 6.35. Oversaturated, pure white pixels are common in the sky and a lighting-corrected color cannot be recovered due to the clipping of the color space.

FIGURE 6.35: Sky color changes on KITTI VO 00 in ca. 2 s. Top to bottom, left to right: Frame 225, 228, 237, 246.

Figure 6.36 shows an example of our reconstruction on KITTI VO 01 using **3DEKF-F** M with GA-Net from far away, so that the skybox can be clearly seen, and the current left stereo frame. Note that parts of the adjacent road are missing in the reconstruction because it is occluded by the highway barrier. Clearly, white and cyan sky pixels have been observed in the previous frames, but are now overwritten by the much darker sky pixels of the current frame. This results in an inconsistently colored sky. Additionally, the semantic segmentation includes false positives at the border, which can be visible in the horizon. For example, the tall beam structure with two attached wire ropes to the right of the road in the bottom image can be seen backprojected to the skybox in the top image along the horizon. Also note that a bright blue rendering background due to the virtual sky instead of black may increase the contrast in stereo matching shadows or holes, which can potentially be an unwanted but salient visual artifact. This can be seen in the top image of Figure 6.36 as we are looking into unobserved scene geometry.

SYNTHIA does not have the problem of different sky color changes. It also features ground truth image segmentation results, but unfortunately only a simple gradient as the sky, *i.e.* without clouds. In Figure 6.37, we show the generated virtual sky on SYNTHIA using the ground truth image segmentation and ELAS as input. The images on the left show the reconstruction with the virtual sky, and the right images show the reconstruction with the alpha channel of the virtual sky colormapped. The black areas in the alpha colormap of the sky are direct observations. We can see that we are indeed able to extrapolate the sky over multiple frames. Looking at the sky directly from a high viewpoint with high pitch (bottom row), we can see that there are again some artifacts in the sky. The wrongly projected vegetation (top of trees) is due to a mismatch of the dataset between depths and the image segmentation, *i.e.* the vegetation is modeled too simplistic. In the corresponding alpha colormap, we can see how scene objects occlude sky observations and how the recent direct



(a) Virtual sky and reconstruction seen from afar.

(b) Current left KITTI VO image.

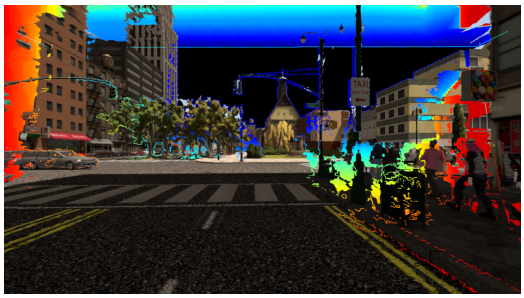
FIGURE 6.36: Inconsistent virtual sky on KITTI VO 01 with some artifacts.

observations (black and dark blue) are propagated to the surrounding area.

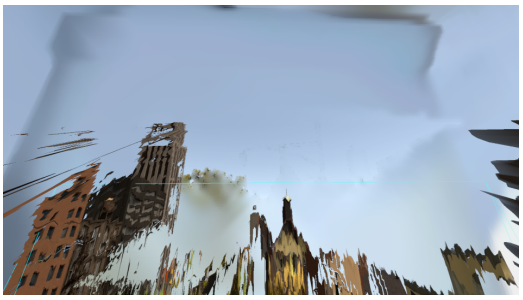
Even though we are able to extrapolate parts of the sky quite well, we observe some limiting factors in this particular approach: First of all, the semantic image segmentation plays a critical role in the sky generation. We effectively extrapolate the border of the observed sky pixels, which means that it is very sensitive to wrong classifications at the semantic segmentation border. SYNTHIA, as a synthetic dataset, is rendered with anti-aliasing measures, resulting in color transitions at depth edges in the scene. But the extracted semantic segmentation labels are discrete classes, which results in some off-color pixels smearing parts of the sky (ghosting artifacts). The bottom left image of Figure 6.37 features a smeared, dark border in the virtual sky because of this.

Here, we used pixel-perfect segmentation results. But for practical purposes, the segmentation results need to be quite accurate to prevent the projection of false structures to the sky. Recent sky segmentation results in broad real-world contexts may not be accurate enough to guarantee high-quality sky generation (see Mihail *et al.* [Mih+16]). To accommodate, it might be necessary to use a morphological operation

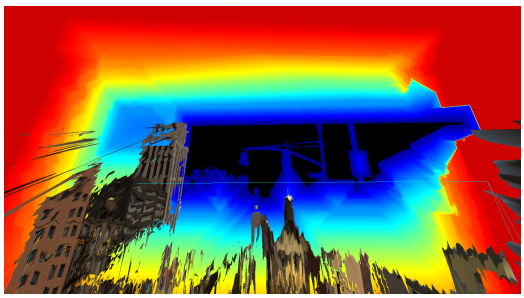
(a) Close to source viewpoint.



(b) Alpha values visualized (source).



(c) High viewpoint with high pitch.



(d) Alpha values visualized (higher, pitched up).

FIGURE 6.37: Sample virtual sky on a snapshot of SYNTHIA-02 reconstructed with ELAS and 3DEKF-F. We visualize the alpha values of the skybox with a colormap from blue to green to red. Observations are in black.

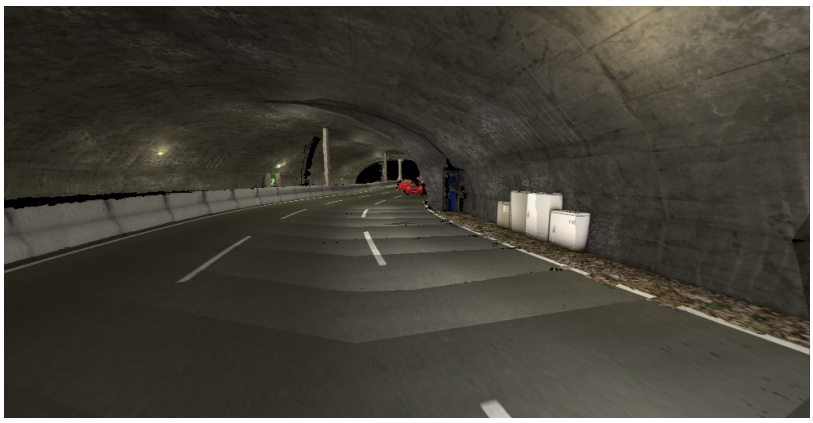
like erosion to focus less on the border pixels of the semantic segmentation and more on the innermost sky pixels, which are more likely to be part of the actual sky. Using alpha matting on the semantic segmentation results of the sky may improve the detection of the actual sky border beforehand.

As an alternative, it might also prove to be more reasonable to simulate a virtual sky and parameterize this sky according to the detected sky pixels. The removal of sky pixels might be a necessary step in each case, so we have detected the sky and have some information about its current appearance. These seed pixels could be used in combination with GPS and time information to show a virtual sun and plausible weather conditions. It might even prove to be enough to just remove the sky pixels and display a regular static skybox, representing the most basic case.

Furthermore, displaying an artificial horizon in the skybox may have beneficial effects for motion sickness according to Diels and Bos [DB16]. As a potential extension, we may limit the sky reconstruction only to the upper hemisphere of the skybox and render a visualization of the horizon explicitly. Further research is needed to determine the extent to which such a visual aid is helpful or disruptive to the immersive experience. In the top left image of Figure 6.37, we can see that the extrapolated sky is shown as background pixels, which might require us to either use more aggressive interpolation to hide stereo matching gaps or to show a differently textured background.

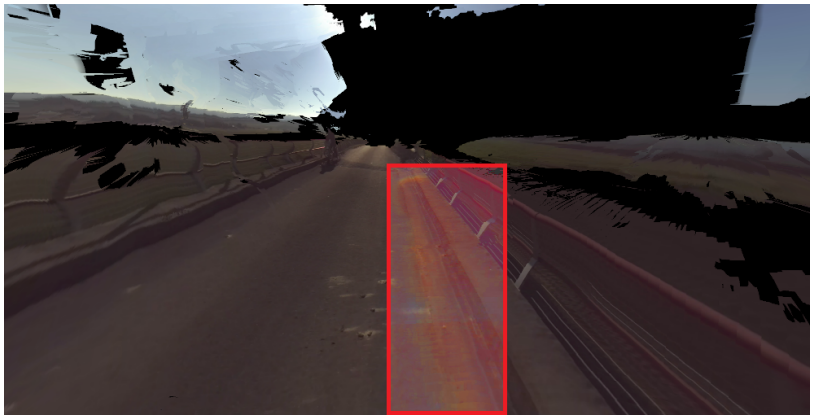
Self-Induced Artifacts

An unavoidable problem for the 3D reconstruction of 3D scenes on real world data is the influence of the observer and its mobile platform. The observer can have no influence on synthetic scenes, but real scenes are influenced for example by the emission of light, shadows, reflections in the scene and artifacts in the camera image. This may cause visible artifacts in the 3D reconstruction over multiple frames.



(a) Car head lights.

(b) Shadow of cyclist with helmet camera.



(c) Smeared lens flare in red box, brightened and saturated color.

FIGURE 6.38: Examples of self-induced reconstruction artifacts over multiple frames of different sequences.

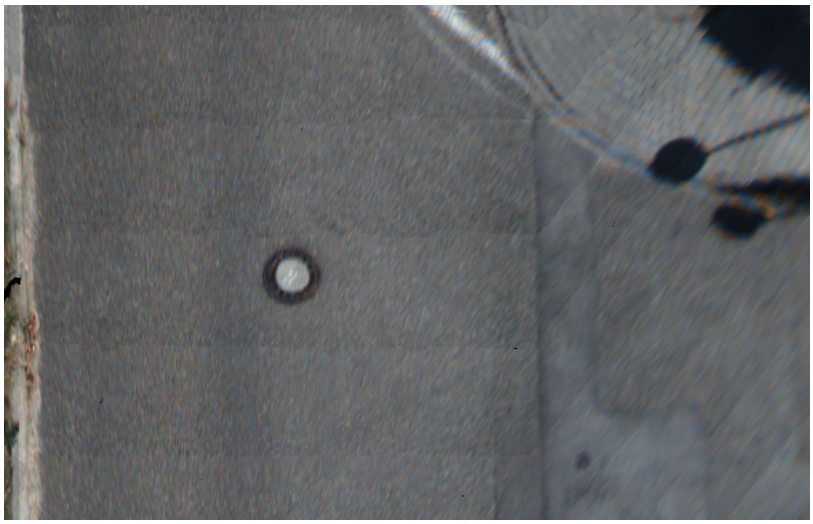
Figure 6.38 shows examples of possible artifacts using our 3DEKF-F M approach on different datasets. The top image shows how the car head lights momentarily affect the road while driving, which is picked up in each tube mesh element. The mid image repeats the shadow of a cyclist with a helmet camera in the same way. Similarly, the camera in the bottom image is now moving towards the sun, which results in the extrusion of a lens flare across the bridge close to the right curb. In an automotive context, lens flares can be further reduced with an integrated lens hood for the stereo camera. But dirt on the windshield can cause backscatter artifacts or parts of the dashboard can be reflected in strong sun light. Visible backscatter artifacts over multiple frames caused by a dust speck can be observed in Figure 6.24 (most prominent on the road in the top image).

Lighting Changes

Conventional stereo algorithms work on images with a bit depth of 8 bit. The KITTI dataset uses an automatic shutter time to adjust to lighting changes because otherwise it could not capture the high dynamic range of changes between *e.g.* shadowy areas and direct sun light exposure. This has visible consequences for the tube mesh: Different stages of lighting changes will be captured discretely over time in separate tube mesh elements. As a result, the tube mesh transitions are very visible due to an increased contrast. This can be clearly seen on the road in Figure 6.34.

In Figure 6.39, we show how a lighting change over a sequence of frames will affect the visual quality compared to static lighting of a road scene from a top-down view. For that, we use our 3DEKF-F M approach on KITTI using GA-Net on KITTI VO 06. Even though the lighting changes only slightly in the top image, the transitions are clearly visible. Whereas in the bottom image, they are barely noticeable. Fortunately, modern vehicle cameras may use high dynamic range technology, which could limit the severity of lighting changes.

As an alternative, one may try to estimate the current lighting per frame according to a physical model to normalize the color channels to a consistent level. This can lead to a consistent virtual sky color and could improve the reconstruction of the tube mesh as well because it makes frame transitions in the textured meshes less obvious. The situation in Figure 6.36 allows us to see the same points in different exposure settings, which may be used to extract high-dynamic range information for these scene parts. Using the brightness estimation (for visual odometry purposes) supplied by [WSC17], we see clear improvements in some part of the scene, but strong deteriorations in other parts of the scene. Generally, the clipped color space in the input images may make this task challenging.



(a) Visible transitions during a small lighting change.

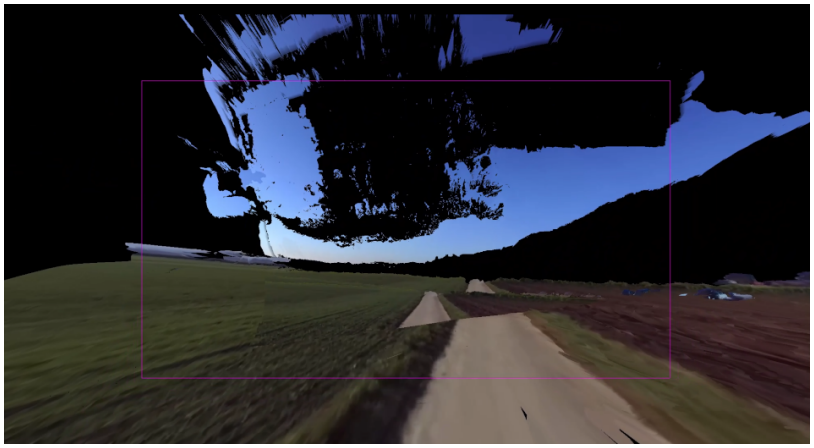
(b) Barely visible transitions during constant lighting.

FIGURE 6.39: 3D Reconstruction of road surfaces with and without lighting changes on the same sequence (KITTI VO 06).

Pose Jumps

Erroneous or inconsistent pose estimates can have dramatic effects on the reconstruction quality. Fortunately, we did not experience large pose jumps in our test sequences. Most small pose inconsistencies (and partly depth inconsistencies) are causing a reconstruction which appears to be choppy since adjacent frames will not align perfectly. The only observed exception for large pose jumps is in our helmet camera sequence. Here, the Stereolabs SDK algorithm may have tried to relocate the camera pose with its SLAM algorithm. These jumps can be detected via a sanity check of the pose consistency. In Figure 6.40, we show two different ways a pose jump could play out using our **3DEKF-F** M approach: In the top image, the camera jumps in such a way that the current reconstruction frame occludes the previous reconstruction. This leads to choppy transitions on the side, but fortunately will still

(a) Current reconstruction cleanly occludes the previous reconstruction.



(b) Previous reconstruction occludes parts of the current reconstruction.

FIGURE 6.40: Pose jumps in our reconstruction as seen from the camera position in the helmet camera sequence with ELAS.

deliver a consistent reconstruction of the current frame. In the bottom image, a worse case is shown. The previous reconstruction is mixed with the current reconstruction (misplaced road segment in the middle) so that some parts of the current reconstruction are occluded by the previous reconstruction due to faulty pose estimates. In both cases we are forced to reset the tube meshes and just show the current frame alone to give the user a less disorienting experience.

Chapter 7

Conclusion and Outlook

7.1 Summary

In this thesis, we addressed the problem of reconstructing urban 3D scenes with a single front stereo over a sequence of frames. The possible novel target applications are, on the one hand, an enhancement of the existing in-car AR system from a 2D overlay to 3D scene content, and, on the other hand, in-car AR and MR applications on HMDs. Providing an immersive experience requires a plausible reconstruction of the scene geometry at high reconstruction and rendering speeds.

For this, we investigated the effects of temporal fusion thoroughly because temporal fusion may provide an additional accuracy increase besides improving geometrical parameters of the stereo setup or the stereo matching itself. We assume that an improved geometrical stereo error will also lead to a better visual quality. We derived an 1D extended information filter and, as an extension, a 3D extended Kalman filter for each individual 3D scene point, which greatly outperforms other naive fusion approaches on simulated stereo measurements. We analyzed multiple theoretical limiting factors for the accuracy of temporal stereo fusion, which may provide new directions for additional research regarding probabilistic filter design.

We successfully integrated the probabilistic filters in a novel point and mesh-based reconstruction pipeline to test temporal fusion on real-world data. The mesh-based pipeline exploits shaders on the GPU to compute the propagation equations of the probabilistic filter to the next frame, which enables an almost real-time processing speed on the used input data. Our image-based mesh rendering concepts allow us to quickly render urban driving scenes with a set of consecutive stereo images and camera poses. On the KITTI VO dataset, we reached a render frame rate of up to 200 Hz, which fulfills our stated goal to achieve modern HMD frame rates. Our novel tube mesh approach for rendering past scene geometry is able to reconstruct the scene with the highest possible and directly observable resolution and allows a reconstruction beyond the current camera frustum for an immersive display of the whole driving scene. Furthermore, we proposed a technique to generate a virtual sky, which can extrapolate detected background sky pixels into unobserved parts of the scene.

We evaluated different reconstruction approaches on real-world and synthetic stereo images in different benchmarks. We were able to substantially improve our image-based stereo metrics results compared to the input with our confidence-based mesh approach at the cost of completeness. Additionally, we highlight challenges in the benchmarking of stereo reconstructions and propose new error metrics and methods to compare the reconstruction results more in-depth. We also propose a view prediction error evaluation for front stereo cameras on a vehicle, in which our full mesh approach vastly outperforms other approaches, which highlights the necessity of a dense scene representation. Furthermore, we showed that we can limit our approach to some extent to further improve its performance without sacrificing quantitative reconstruction quality. In the end, we demonstrated our approach on various datasets including real-world data with small stereo baselines and even in a cycling context. We conclude with an extensive analysis of the observed limitations of the used setup and approaches, highlighting additional research potential.

7.2 Future Work

For future work, we strive to implement a real prototype of our proposed system for AR and MR. For this, it might be worthwhile at first to replace our fusion with the simplistic **Replace** variant in a mesh-based reconstruction pipeline to save processing speed because the stereo matching and reconstruction speed still pose real-time bottlenecks. However, not every frame needs to be reconstructed for a dense visualization and some latency might be tolerable. This would require a smooth fade-in concept when integrating new frames live. We also expect that semantic segmentation will be available in modern cars soon thanks to driver assistance systems, which can be leveraged to remove dynamic objects when adding a new tube mesh to the reconstruction. We also fully expect that future deep learning approaches may improve the visual fidelity of the scene even more and may also be directly leveraged to improve the render from a different camera viewpoint, including estimating the 3D scene content of unobserved areas.

For an immersive experience, the sky pixels need to be removed from the reconstruction and require a plausible replacement. Sky detection “in the wild” still requires more research and a suitable replacement of the sky in the automotive context may also be further explored. A more aggressive interpolation of the background may prove to be worthwhile to hide reconstruction holes, which would show parts of the virtual sky.

Another possible extension would be to replace the implicit scene flow provided by warping pixels to the new frame using the estimated camera poses. This can be achieved by an explicit dense scene flow algorithm. This may be advantageous because dynamic objects can have correct correspondences between frames compared to the implicit scene flow method. Alternatively, it might be interesting to analyze the effects that the quality of the pose estimation itself or scene flow have on the

achievable stereo accuracy using temporal fusion, *i.e*. our algorithms. This would allow us to reason for example whether we need a certain (scene dependent) pose accuracy to achieve a certain 3D reconstruction improvement or when temporal fusion might fail.

Regarding the use of probabilistic filters, we think that some of the limitations, like temporal error correlation, are hard to alleviate: The similarity of image patches between frames is evidence of a good temporal correspondence, but it is also harbors the potential that the stereo matching error of these patches might be correlated, *i.e*. resulting in a similar local disparity error. Still, it might be interesting to see if a probabilistic filter that incorporates the actual fat-tailedness of our empirical disparity error distribution can improve fusion results or if the true disparity error distribution can be derived analytically and simulated. Additionally, the disparity error dependency on the true disparity itself (for high disparities) may be further analyzed and incorporated in a filter model.

List of Abbreviations

ALC	Reconstruction approach of **Alc**antarilla *et al.* [ABD13]
AR	**A**ugmented **R**eality
BG	**B**ack**g**round
CDF	**C**umulative **D**istribution **F**unction
DoF	**D**egrees **o**f **F**reedom
DR	**D**iminished **R**eality
EIF	**E**xtended **I**nformation **F**ilter
EKF	**E**xtended **K**alman **F**ilter
ELAS	**E**fficient **La**rge-scale **S**tereo Matching [GRU10]
FG	**F**ore**g**round
FoV	**F**ield **o**f **V**iew
FSC	**F**ree**s**pace **C**heck
GA-Net	**G**uided **A**ggregation **Net** [Zha+19]
GPU	**G**raphics **P**rocessing **U**nit
HCI	**H**eidelberg **C**ollaborary for **I**mage Processing (stereo metrics)
HUD	**H**ead-**u**p **D**isplay
HMD	**H**ead-**m**ounted **D**isplay
IMU	**I**nertial **M**easurement **U**nit
IDW	**I**nverse **D**istance **W**eighting (binning approach)
Lidar	**Li**ght **D**etection **a**nd **R**anging
MR	**M**ixed **R**eality
MVS	**M**ulti-**v**iew **S**tereo
NCC	**N**ormalized **c**ross-**c**orrelation
NN	**N**earest **N**eighbor (binning approach)
PDF	**P**robability **D**ensity **F**unction
PSMNet	**P**yramid **S**tereo **M**atching **Net**work [CC18]
PSW	**P**ixel **S**ize **W**eighting (binning approach)
RMSE	**R**oot-**m**ean-**s**quare-**e**rror
SfM	**S**tructure **f**rom **M**otion
SLAM	**S**imultaneous **L**ocalization **a**nd **M**apping
SS	**S**tereo**S**can reconstruction approach [GZS11]
SIDW	**S**ticky **I**nverse **D**istance **W**eighting (binning approach)
SIMD	**S**ingle **i**nstruction, **m**ultiple **d**ata
SSIM	**S**tructural **Sim**ilarity Index
TSDF	**T**runcated **S**igned **D**istance **F**unction

VO **V**isual **O**dometry
VR **V**irtual **R**eality

List of Figures

List of Tables

Bibliography

[ABD13] Pablo F. Alcantarilla, Chris Beall, and Frank Dellaert. "Large-scale Dense 3D Reconstruction From Stereo Imagery". In: Georgia Institute of Technology. 2013.

[AZD13] Dimitrios S Alexiadis, Dimitrios Zarpalas, and Petros Daras. "Real-time, Full 3-D Reconstruction of Moving Foreground Objects from Multiple Consumer Depth Cameras". In: *IEEE Transactions on Multimedia* 15.2 (2013), pp. 339–358.

[Bar17] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[Bâr+18] Ioan Andrei Bârsan et al. "Robust Dense Mapping for Large-Scale Dynamic Environments". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7510–7517.

[BH87] Steven D Blostein and Thomas S Huang. "Error Analysis in Stereo Determination of 3-D Point Positions". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 6 (1987), pp. 752–765.

[Bla14] Jose Luis Blanco. *nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) wih KD-trees*. https://github.com/jlblancoc/nanoflann. 2014.

[Ble18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam, 2018. URL: http://www.blender.org.

[Box+15] George EP Box et al. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.

[CBM17] Cevahir Cigla, Roland Brockers, and Larry Matthies. "Gaussian Mixture Models for Temporal Depth Fusion". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 889–897.

[CC18] Jia-Ren Chang and Yong-Sheng Chen. "Pyramid Stereo Matching Network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5410–5418.

[Che+17] Liang-Chieh Chen et al. "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[CL96] Brian Curless and Marc Levoy. "A Volumetric Method for Building Complex Models From Range Images". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM. 1996, pp. 303–312.

[Con18] Continental AG. *Mehr Sicherheit durch freie Sicht: Continental behebt mit Virtueller A-Säule tote Winkel im vorderen Blickfeld*. https://www.continental.com/de/presse/pressemitteilungen/2018-10-25-virtual-a-pillar-148412. Accessed: 2019-09-05. Oct. 2018.

[Cor+08] Nico Cornelis et al. "3D Urban Scene Modeling Integrating Recognition and Reconstruction". In: *International Journal of Computer Vision* 78.2-3 (2008), pp. 121–141.

[Dai19] Daimler AG. *Mercedes Benz EQC: MBUX Augmented Reality für Navigation*. https://www.mercedes-benz.de/passengercars/mercedes-benz-cars/models/eqc/comfort.pi.html/mercedes-benz-cars/models/eqc/comfort/comfort-gallery/augmented-video. Accessed: 2019-08-30. 2019.

[DB16] Cyriel Diels and Jelte E Bos. "Self-driving Carsickness". In: *Applied Ergonomics* 53 (2016), pp. 374–382.

[DiV07] Stephen DiVerdi. *Towards anywhere augmentation*. University of California, Santa Barbara, 2007.

[Dod04] Neil A Dodgson. "Variation and Extrema of Human Interpupillary Distance". In: *Stereoscopic Displays and Virtual Reality Systems XI*. Vol. 5291. International Society for Optics and Photonics. 2004, pp. 36–46.

[Dum15] Maarten Dumont. "Real-Time View Interpolation for Eye Gaze Corrected Video Conferencing". PhD thesis. Hasselt University, 2015.

[Eat+16] John W. Eaton et al. *GNU Octave Version 4.2.0 Manual: A High-level Interactive Language for Numerical Computations*. http://www.gnu.org/software/octave/doc/interpreter. 2016.

[EMW02] Geoffrey Egnal, Max Mintz, and Richard P. Wildes. "A Stereo Confidence Metric Using Single View Imagery". In: *Proc. Vision Interface*. Citeseer. 2002, pp. 162–170.

[ESC15] Jakob Engel, Jörg Stückler, and Daniel Cremers. "Large-scale Direct SLAM with Stereo Cameras". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 1935–1942.

[Fac19a] Facebook Technologies, LLC. *Oculus Quest: All-in-One-VR-Headset*. https://www.oculus.com/quest/. Accessed: 2019-10-02. 2019.

[Fac19b] Facebook Technologies, LLC. *Oculus Rift: VR Headset for VR Ready PCs*. https://www.oculus.com/rift/. Accessed: 2019-11-05. 2019.

[Fra+05] Uwe Franke et al. "6D-Vision: Fusion of Stereo and Motion for Robust Environment Perception". In: *Joint Pattern Recognition Symposium*. Springer. 2005, pp. 216–223.

[Gal11] David Gallup. "Efficient 3D Reconstruction of Large-scale Urban Environments from Street-level Video". PhD thesis. The University of North Carolina at Chapel Hill, 2011.

[Gey+06] Christopher Geyer et al. "The Recursive Multi-Frame Planar Parallax Algorithm". In: *Third International Symposium on 3D Data Processing, Visualization, and Transmission*. IEEE. 2006, pp. 17–24.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[GRU10] Andreas Geiger, Martin Roser, and Raquel Urtasun. "Efficient Large-Scale Stereo Matching". In: *Asian Conference on Computer Vision*. Springer. 2010, pp. 25–38.

[Grü13] Christian Grünler. *Augmented Reality: Von der Vision zur Realität*. https://blog.daimler.com/2013/10/14/augmented-reality-von-der-vision-zur-realitaet/. Accessed: 2019-06-04. Oct. 2013.

[GZS11] Andreas Geiger, Julius Ziegler, and Christoph Stiller. "StereoScan: Dense 3D Reconstruction in Real-time". In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 963–968.

[Hae+18] Jonas Haeling et al. "In-Car 6-DoF Mixed Reality for Rear-Seat and Co-Driver Entertainment". In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. 2018, pp. 757–758.

[Har+14] Hannes Harms et al. "Accuracy Analysis of Surface Normal Reconstruction in Stereo Vision". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE. 2014, pp. 730–736.

[Her08] Marion A Hersh. "Perception, the Eye and Assistive Technology Issues". In: *Assistive technology for visually impaired and blind people*. Springer, 2008, pp. 51–101.

[Hir08] Heiko Hirschmüller. "Stereo Processing by Semiglobal Matching and Mutual Information". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 328–341.

[HK12] Ralf Haeusler and Reinhard Klette. "Analysis of KITTI Data for Stereo Analysis with Stereo Confidence Measures". In: *European Conference on Computer Vision*. Springer. 2012, pp. 158–167.

[HM12] Xiaoyan Hu and Philippos Mordohai. "A Quantitative Evaluation of Confidence Measures for Stereo Vision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2121–2133.

[HNK13] Ralf Haeusler, Rahul Nair, and Daniel Kondermann. "Ensemble Learning for Confidence Measures in Stereo Vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 305–312.

[HNS19a] Jonas Haeling, Marc Necker, and Andreas Schilling. *Calibrating Depth Sensors with a Genetic Algorithm*. Tech. rep. Universität Tübingen, 2019.

[HNS19b] Jonas Haeling, Marc Necker, and Andreas Schilling. "Towards Immersive Stereo Vision From a Mobile Platform". In: *Eleventh International Conference on Machine Vision (ICMV 2018)*. Ed. by Antanas Verikas et al. Vol. 11041. International Society for Optics and Photonics. SPIE, 2019, pp. 222 –229. DOI: 10.1117/12.2522772. URL: https://doi.org/10.1117/12.2522772.

[HNS20] Jonas Haeling, Marc Necker, and Andreas Schilling. "Dense Urban Scene Reconstruction using Stereo Depth Image Triangulation". In: *Twelfth International Conference on Machine Vision (ICMV 2019)*. Vol. 11433. International Society for Optics and Photonics. SPIE, 2020, pp. 483 –490. DOI: 10.1117/12.2556688. URL: https://doi.org/10.1117/12.2556688.

[Hob15] Benedict Hobson. *MINI's augmented-reality glasses can make cars transparent*. https://www.dezeen.com/2015/04/24/mini-augmented-reality-glasses-allow-drivers-to-see-through-the-body-of-their-car/. Accessed: 2019-10-11. Apr. 2015.

[Hoc+17] Philipp Hock et al. "CarVR: Enabling In-Car Virtual Reality Entertainment". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. New York, NY, USA: ACM, 2017, pp. 4034–4044. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025665. URL: http://doi.acm.org/10.1145/3025453.3025665.

[hol19] holoride GmbH. *Holoride*. https://www.holoride.com/. Accessed: 2019-16-10. 2019.

[Hon19] Katrin Honauer. "Performance Metrics and Test Data Generation for Depth Estimation Algorithms". PhD thesis. University of Heidelberg, 2019.

[Käh+15] Olaf Kähler et al. "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices". In: *IEEE Transactions on Visualization and Computer Graphics* 21.11 (2015), pp. 1241–1250.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson Surface Reconstruction". In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 2006.

[Kel+13] Maik Keller et al. "Real-Time 3D Reconstruction in Dynamic Scenes using Point-based Fusion". In: *International Conference on 3D Vision*. IEEE. 2013, pp. 1–8.

[Kod+17] R. Kodama et al. “COMS-VR: Mobile Virtual Reality Entertainment System using Electric Car and Head-mounted Display”. In: *2017 IEEE Symposium on 3D User Interfaces (3DUI)*. 2017, pp. 130–133. DOI: 10.1109/3DUI.2017.7893329.

[KS02] Hyung Woo Kang and Sung Yong Shin. “Tour Into the Video: Image-based Navigation Scheme for Video Sequences of Dynamic Scenes”. In: *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM. 2002, pp. 73–80.

[Lan14] Land Rover USA. *Discovery Vision Concept 'Transparent Hood' Technology*. https://www.youtube.com/watch?v=L7j1daOk72c. Accessed 2019-10-11. Apr. 2014.

[LK90] Sukhan Lee and Youngchul Kay. “A Kalman Filter Approach for Accurate 3-D Motion Estimation from a Sequence of Stereo Images”. In: *10th International Conference on Pattern Recognition, 1990. Proceedings.* Vol. 1. IEEE. 1990, pp. 104–108.

[Ma+18] Hao Ma et al. “Confidence-based Iterative Efficient Large-scale Stereo Matching”. In: *Cogent Engineering* 5.1 (2018), p. 1427676.

[MAT17] Raul Mur-Artal and Juan D Tardós. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[MB17] Mark McGill and Stephen A. Brewster. “I Am The Passenger: Challenges in Supporting AR/VR HMDs In-Motion”. In: *Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications Adjunct*. AutomotiveUI '17. New York, NY, USA: ACM, 2017, pp. 251–251. ISBN: 978-1-4503-5151-5. DOI: 10.1145/3131726.3131876. URL: http://doi.acm.org/10.1145/3131726.3131876.

[MG15] Moritz Menze and Andreas Geiger. “Object Scene Flow for Autonomous Vehicles”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3061–3070.

[MG94] Larry Matthies and Pierrick Grandjean. “Stochastic Performance, Modeling and Evaluation of Obstacle Detectability with Imaging Range Sensors”. In: *IEEE Transactions on Robotics and Automation* 10.6 (1994), pp. 783–792.

[Mic19] Microsoft. *Microsoft HoloLens | Mixed Reality for Business*. https://www.microsoft.com/en-us/hololens. Accessed: 2019-10-02. 2019.

[Mih+16] Radu P Mihail et al. “Sky Segmentation in the Wild: An Empirical Study”. In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–6.

[Mil+95] Paul Milgram et al. "Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum". In: *Telemanipulator and telepresence technologies*. Vol. 2351. International Society for Optics and Photonics. 1995, pp. 282–292.

[MIS17] Shohei Mori, Sei Ikeda, and Hideo Saito. "A Survey of Diminished Reality: Techniques for Visually Concealing, Eliminating, and Seeing Through Real Objects". In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), pp. 1–14.

[MK13] Sandino Morales and Reinhard Klette. "Kalman-filter Based Spatio-temporal Disparity Integration". In: *Pattern Recognition Letters* 34.8 (2013), pp. 873–883.

[MKS89] Larry Matthies, Takeo Kanade, and Richard Szeliski. "Kalman Filter-based Algorithms for Estimating Depth from Image Sequences". In: *International Journal of Computer Vision* 3.3 (1989), pp. 209–238.

[ML04] Don Murray and James J Little. "Environment Modeling with Stereo Vision". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 3116–3122.

[MSD07] Matthias Müller, Simon Schirm, and Stephan Duthaler. "Screen Space Meshes". In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 2007, pp. 9–15.

[Mus+13] Przemyslaw Musialski et al. "A Survey of Urban Reconstruction". In: *Computer graphics forum*. Vol. 32. 6. Wiley Online Library. 2013, pp. 146–177.

[New+11] Richard A Newcombe et al. "KinectFusion: Real-Time Dense Surface Mapping and Tracking". In: *10th IEEE International Symposium on Mixed and Augmented reality (ISMAR)*. IEEE. 2011, pp. 127–136.

[Ope19] OpenMP. *Home - OpenMP*. https://www.openmp.org/. Accessed: 2019-12-04. 2019.

[PGS13] David Pfeiffer, Stefan Gehrig, and Nicolai Schneider. "Exploiting the Power of Stereo Confidences". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 297–304.

[Pir+18] Taihú Pire et al. "Real-time Dense Map Fusion for Stereo SLAM". In: *Robotica* 36.10 (2018), pp. 1510–1526.

[Pla19] Lennart Alexander Planz. "Visuelle Odometrie für Augmented Reality im Fahrzeug". German text. MA thesis. Technische Universität Illmenau, 2019.

[Pol+08] Marc Pollefeys et al. “Detailed Real-time Urban 3D Reconstruction from Video”. In: *International Journal of Computer Vision* 78.2-3 (2008), pp. 143–167.

[Pol+15] Marc Pollefeys et al. *Urban 3D Modelling from Video*. http://www.cs.unc.edu/Research/urbanscape/. Accessed: 2019-09-30. 2015.

[PRL16] Sudeep Pillai, Srikumar Ramalingam, and John J Leonard. “High Performance and Tunable Stereo Reconstruction”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3188–3195.

[PY15] Min-Gyu Park and Kuk-Jin Yoon. “Leveraging Stereo Matching with Learning-based Confidence Measures”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 101–109.

[Ram+16] F. Rameau et al. “A Real-Time Augmented Reality System to See-Through Cars”. In: *IEEE Transactions on Visualization and Computer Graphics* 22.11 (Nov. 2016), pp. 2395–2404. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2593768.

[RFM17] Andrea Romanoni, Daniele Fiorenti, and Matteo Matteucci. “Mesh-based 3D Textured Urban Mapping”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3460–3466.

[Ros+16] German Ros et al. “The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Sch+14] Daniel Scharstein et al. “High-resolution Stereo Datasets with Subpixel-accurate Ground Truth”. In: *German conference on pattern recognition*. Springer. 2014, pp. 31–42.

[Sch+15] Thomas Schöps et al. “3D Modeling on the Go: Interactive 3D Reconstruction of Large-scale Scenes on Mobile Devices”. In: *2015 International Conference on 3D Vision (3DV)*. IEEE. 2015, pp. 291–299.

[Sch18] Stephan Schmid. “Semi-Dense Filter-Based Visual Odometry for Automotive Augmented Reality Applications”. PhD thesis. University of Stuttgart, 2018.

[Sch99] Daniel Scharstein. *View Synthesis using Stereo Vision*. Springer-Verlag, 1999.

[Sen+13] Sunando Sengupta et al. “Urban 3D Semantic Modelling Using Stereo Vision”. In: *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 580–585.

[SG14] Miriam Schönbein and Andreas Geiger. “Omnidirectional 3D Reconstruction in Augmented Manhattan Worlds”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 716–723.

[SL00] Nicu Sebe and Michael S Lew. "Maximum Likelihood Stereo Matching". In: *Proceedings of the 15th International Conference on Pattern Recognition*. Vol. 1. IEEE. 2000, pp. 900–903.

[SMS07] Gabe Sibley, Larry Matthies, and Gaurav Sukhatme. "Bias Reduction and Filter Convergence for Long Range Stereo". In: *Robotics Research* (2007), pp. 285–294.

[SP16] Akihito Seki and Marc Pollefeys. "Patch Based Confidence Prediction for Dense Disparity Map". In: *BMVC*. Vol. 2. 3. 2016, p. 4.

[SS02] Daniel Scharstein and Richard Szeliski. "Middlebury Stereo Vision Page". In: *Online at* `http://vision.middlebury.edu/stereo/eval3/` Version 3 (2002).

[SSM06] Gabe Sibley, Gaurav S Sukhatme, and Larry H Matthies. "The Iterated Sigma Point Kalman Filter with Applications to Long Range Stereo." In: *Robotics: Science and Systems*. Vol. 8. 1. 2006, pp. 235–244.

[Sta18] Tobias Staib. "Rekonstruktion und Visualisierung von urbanen Szenen fur Augmented Reality Navigationssysteme". MA thesis. Hochschule der Medien Stuttgart, 2018.

[Ste19] Stereolabs. *Zed Mini Stereo Camera - Stereolabs*. `https://www.stereolabs.com/zed-mini/`. Accessed: 2019-11-28. 2019.

[Tan+16] Michael Tanner et al. "DENSER Cities: A System for Dense Efficient Reconstructions of Cities". In: *arXiv preprint arXiv:1604.03734* (2016).

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[Tol+09] Engin Tola et al. *Virtual View Generation with a Hybrid Camera Array*. Tech. rep. 2009.

[Tos+18] Fabio Tosi et al. "Beyond Local Reasoning for Stereo Confidence Estimation with Deep Learning". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 319–334.

[Use+15] Vladyslav Usenko et al. "Reconstructing Street-Scenes in Real-Time from a Driving car". In: *Proceedings of the 2015 International Conference on 3D Vision*. IEEE. 2015, pp. 607–614.

[Val18] Valeo Group. *Valeo XtraVue, see through cars*. `https://www.youtube.com/watch?v=F4-wWfCcyK4`. Accessed: 2019-10-11. July 2018.

[Van+17] Kenneth Vanhoey et al. "VarCity - the Video: The Struggles and Triumphs of Leveraging Fundamental Research Results in a Graphics Video Production". In: *ACM SIGGRAPH 2017 Talks*. ACM. 2017, p. 48.

[Wae+17] Michael Waechter et al. "Virtual Rephotography: Novel View Prediction Error for 3D Reconstruction". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), 45a.

[Wal15] Wall Street Journal. *Virtual Reality and Real-Life Racing Combine in Driving First*. https://youtu.be/73Qmg_7Nx98. Accessed: 2019-10-02. Dec. 2015.

[Wan+04] Zhou Wang et al. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[WC11] Andreas Wedel and Daniel Cremers. *Stereo Scene Flow for 3D Motion Analysis*. Springer Science & Business Media, 2011.

[WGS19] Kaixuan Wang, Fei Gao, and Shaojie Shen. "Real-time Scalable Dense Surfel Mapping". In: *The 2019 International Conference on Robotics and Automation (ICRA)*. 2019.

[WSC17] Rui Wang, Martin Schworer, and Daniel Cremers. "Stereo DSO: Large-scale Direct Sparse Visual Odometry with Stereo Cameras". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3903–3911.

[Yos+08] Takumi Yoshida et al. "Transparent Cockpit: Visual Assistance System for Vehicle using Retro-reflective Projection Technology". In: *2008 IEEE Virtual Reality Conference*. IEEE. 2008, pp. 185–188.

[Zha+15] Chi Zhang et al. "MeshStereo: A Global Stereo Model With Mesh Alignment Regularization for View Interpolation". In: *The IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 2057–2065.

[Zha+19] Feihu Zhang et al. "GA-Net: Guided Aggregation Net for End-to-end Stereo Matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 185–194.

[Zit+] C. Lawrence Zitnick et al. "High-quality Video View Interpolation Using a Layered Representation". In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. ACM, pp. 600–608.