Okan Erat

Image-Based Modeling and Rendering for Telepresence in Remote Exploration Scenarios

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Dieter Schmalstieg

Institute of Computer Graphics and Vision

Graz, Austria, November 2020

Abstract

Remote exploration, also known as tele-exploration, is an important application scenario of telepresence. It allows users to discover a remote environment, at the comfort of their own place and securely away from potential harms of the remote location. The range of applications can stretch from surveillance and rescue operations to space exploration. However, being remote to an environment, but yet still performing a task as if present, is a challenging problem. In order to ease the problem of tele-exploration and to reduce a user's cognitive load, in this thesis, we introduce a set of visualization and interaction methods.

We present a photo-realistic rendering method by adapting and improving Image-Based Modeling and Rendering (IBMR) techniques for real-time performance. Specifically, we demonstrate a novel view planning method for real-time memory management. In addition, we improve the rendering quality with our spatio-temporally coherent inpainting method. We inpaint 3D regions where a 3D reconstruction system couldn't perform as intended and created holes in the model.

Despite having a crucial role in remote exploration, visualization alone is not sufficient to address the needs of every tele-exploration scenario. For scenarios where interactions are also desirable, on top of our *IBMR* system, we showcase our novel interaction techniques with drones.

Kurzfassung

Fernerkundung, auch bekannt als Tele-Erkundung, ist ein wichtiges Anwendungsszenario von Telepräsenz.

Es erlaubt Anwendern eine entfernte Umgebung wahrzunehmen, ohne sich dabei eventuellen Sicherheitsrisiken an einem entfernten Ort auszusetzen. Darüber hinaus kann der entfernte Ort von überall aus erkundet werden, wie zum Beispiel bequem vom eigenen Heim aus.

Das potentielle Spektrum von Anwendungen reicht von Überwachung und Rettungsoperationen bis hin zur Erkundung des Weltraums. Eine Aufgabe mittels Telepresenz durchzuführen, ohne dabei selber physisch am Zielort anwesend zu sein, stellt allerdings eine Herausforderung dar. Diese Arbeit stellt verschiedene Visualisierungs- und Interaktionsmethoden vor, die der Verringerung der kognitiven Belastung des Anwenders bei der Durchführung einer solchen Tele-Operation dienen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

The text document uploaded to TUGRAZonline is identical to the presented doctoral thesis.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Ort	Datum	Unterschrift

Acknowledgments

First and foremost I would like to thank Prof. Dr. Dieter Schmalstieg for giving me the opportunity and financial support to write my thesis with him. Throughout my thesis, his continued guidance and support made this thesis possible. Also I would like to thank him for giving me the freedom to choose my research topics. A special thanks goes to Dr. Denis Kalkofen for always providing me with needed related work, no matter what topic I was talking about. His support during my first publication played a crucial role in my PhD.

I also would like to thank Dr. Peter Roth and Dr. Clemens Arth for supporting me with all my computer vision related questions.

Most importantly, this thesis wouldn't have been possible without continued motivational support of my family: Seyit Erat, Elif Erat, Ozan Erat. Not to forget my in-law Dr. Melissa Erat and dearest nephews Miles and Mus.

Finally, I would like to thank my colleagues Dr. Shohei Mori, Markus Hoell, Dr. Peter Mohr, David Mandl and Dr. Christian Pirchheim for their collaboration.

Contents

1 Introduction

Contents

1.1 Telepresence

Telepresence allows a person to virtually experience a feeling of presence at a remote location by utilizing a set of technologies such as robotics, computer graphics and computer vision. A precise definition and quantification of sensory feedback required to be considered telepresent somewhere is still under discussion by the scientific community.

Telepresence has a range of applications that can even include live television broadcasting, phone or video conferencing, depending on the definition. Therefore, this thesis assumes a stricter definition and expects at least visual sensory information to be present from a distant location with the flexibility of changing the viewpoint.

Figure 1.1: Major milestones in Telepresence.© Dan McCoy/Rainbow, © Argonne National Laboratory.

1.2 A Brief History of Telepresence

Potential applications of telepresence are not only evaluated by scholars, but started to be conceptualized much earlier by the science fiction community, including novelists and movie makers (Figure 1.1). The earliest mention of a telepresence system can be attributed to the novel titled Waldo. Written by Robert A. Heinlein in 1942, Waldo presents the fundamental idea of a telepresence system. In the novel, the main character who is living off earth and wearing special hand gloves, trains and controls factory workers on the earth via the counterparts of the hand gloves. The earliest implementation of the idea of Waldo can be traced back to 1949, to the robotist Raymond Goertz. Goertz developed the first mechanical slave-master manipulator in order to allow scientist to safely experiment on nuclear materials behind protected walls and glass [39]. This mechanical coupling of hand motions with the distant actuator created a major milestone in force feedback sensation, therefore this work differentiates itself from teleoperation. Later, Goertz changed his mechanical design and coupled a master-slave system with electrical communication, where both sides had a sensory mechanism. By 1969, the author Fred Saberhagen described the complete idea of a telepresence system in his novel Brother Assassin. Saberhagen describes a full robotic body that is controlled immersively by a remote operator. Despite the wide range of existing applications, it wasn't until 1980 that the word telepresence was used. The term is first coined by the American scientist Marvin Minsky in an effort to emphasize the immersive nature of this new technology and to differentiate it from remote control tools known as teleoperator and telefactor.

In contrast to teleoperation, telepresence may or may not involve physical interaction with the remote environment, while it must receive at least one sensory information from a remote location.

In 1992, the first complete telepresence system was developed by the US Air Force, known as virtual fixtures [113]. It is considered complete, as it involved all sensory inputs; sound, image, force-feedback from remote location. Thanks to the developments in visual computing and increasing computation power, by the early 2000s, it became possible to reconstruct the remote environment and render it from a desired viewpoint of the user [100, 101, 120]. The user's viewpoint became independent of the remote camera, and this eliminated motion sickness related to network lag. As a further development, some research work also involved Image-Based Modeling and Rendering (IBMR) techniques to create a more realistic telepresence experience [103]. However due to memory and computation demand, previous to the writing of this thesis, *IBMR* techniqes were never applied for a tele-exploration task, which is an application of telepresence.

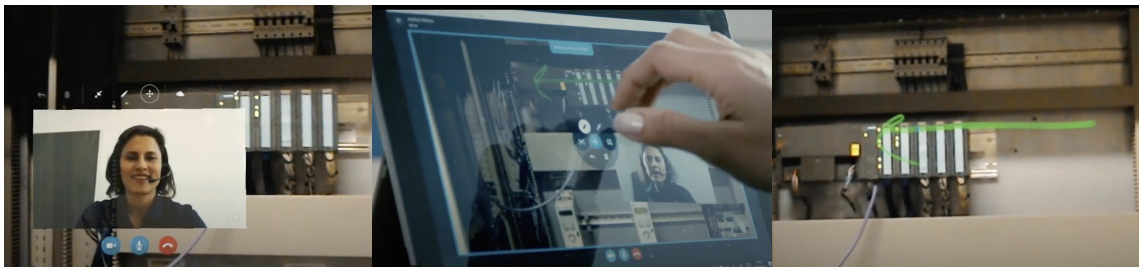


Figure 1.2: Andritz remote assistance tool. © Andritz AG

1.3 Tele-Exploration

As a special application of telepresence, tele-exploration allows users to remotely discover an environment with a feeling of presence and flexibility to change their viewpoint in the environment.

Traditionally, data is collected from the remote enviroment progressively via a remote controlled robot, just like in a teleoperation task. Tele-operated robots can easily access locations which are impassable or too dangerous for humans to reach. Using remotely operated robots allows exploring such areas from a safe distance, providing essential data for diverse applications, such as rescue missions, infrastructure inspection or just photographic exploration.

In contrast to teleoperation, tele-exploration also allows data to be collected using any kind of camera device operated in the remote location by a person. Therefore, it can be used for remote assistance applications or simply coordinating a group of workers in a remote factory (see Figure 1.2).

1.4 Problem Statement

Tele-exploration tasks, just like any other telepresence application, can be implemented with or without granting the remote operator physical interaction abilities. Hence, this thesis will also consider exploration scenarios where physical interaction could be required. In particular, we identified 2 main problems with in the remote exploration problem: (1) Improving and adapting *IBMR* systems for the requirements of the exploration task; (2) assisting interactions with *IBMR*.

1.4.1 Improving and Adapting *IBMR* Systems for Tele-Exploration

Advanced Simultaneous Localization and Mapping (SLAM) can obtain geometric reconstructions in real time. In particular, mobile RGB-D sensors make it easy to obtain dense geometry. The

reconstructed scene can be rendered in real-time with *free viewpoint* control by the user. Free-viewpoint rendering can instantly be used on incomplete scenes, even while the reconstruction is still ongoing. Apart from feedback for the camera operator, instant reconstruction enables a tele-exploration system to be free from the on-board camera's viewpoint [117].

However, instant reconstruction is still an emerging technology and requires further technical improvements. Among these required improvements of tele-exploration enabling technology, we have identified two areas that we consider particularly important:

(1) *Photometric reconstruction.* Acquiring surface colors and textures is often only treated as a byproduct of geometric reconstruction. A typical approach storing only a single, averaged color per scene point cannot preserve high-quality scene appearance.

(2) *Spatio-temporally coherent inpainting.* Novel views generated from incomplete scenes can contain seams and holes. Artifact-free, dense scanning can be difficult and time-consuming, if the scene contains hard-to-reach locations. Moreover, users of tele-exploration applications may want to remove objects or replace them with others. For example, a user of an interior design application may wish to try out new furniture without having to physically remove existing furniture first [140]. In these cases, we would like a spatiotemporally coherent inpainting to fill in the blanks or unwanted areas.

Many techniques in these areas exist, but most are designed for *offline* use. Approaches that run for minutes or hours [4, 6, 48, 133] and require unbounded memory are not compatible with tele-exploration requirements. In particular, global methods depend on scene complexity, making it extremely difficult to maintain strictly bounded computation times and memory use.

1.4.2 Assisting Interactions with *IBMR*

Discovering a remote environment is a complex and sometimes a dangerous task, which often involves teleoperation robots. Introducing *IBMR* into this particular task allow remote operators to control the robot from their own independent viewpoint, while maintaining the telepresence feeling thanks to *IBMR*.

Depending on the situation, a tele operator may choose between two principal modes of control. In either mode, a conventional handheld controller is used to navigate the robot, but the viewing differs between modes: In *exocentric viewing* mode, the operator observes the robot from a stationary viewpoint while steering. In *egocentric viewing* – or first-person – mode, video from the robot's on-board camera is streamed to the operator to inform the steering. Recently, wearing a Head-Mounted

Display (HMD) to watch the streaming video has become a popular enhancement of the egocentric mode.

Obviously, controlling a robot in exocentric mode is difficult in the presence of significant occlusions, and it becomes impossible when the robot is exploring the inside of a building. In this case, the operator is forced to use an egocentric mode based on streaming the image from the on-board camera. But navigation in a narrow environment, while relying exclusively on an on-board camera with a potentially limited field of view, can be difficult.

In such a difficult situation, the operator may substantially benefit from an autopilot system, which enables indirect control. The operator specifies a destination, and the autopilot steers the robot there autonomously. State-of-the-art autopilots stabilize the robot's pose during motion and prevent crashing into obstacles, but cannot perform path-planning or way-finding. The operator must still maintain the overview and guide the robot step by step.

Precisely this overview is lacking if only an egocentric view is available, making robot control more difficult than necessary. An exocentric view, showing the robot's surrounding from the operator's rather than the robot's point of view, would clearly be preferable.

Virtual Reality (VR) can provide a synthetic exocentric view by combining the live video with a 3D model of the occluded environment from any viewpoint, independent of the user's physical viewpoint. Using image-based rendering of the robot's video stream delivers a realistic impression with live updates [99].

In addition to supporting a virtual viewpoint, Augmented Reality (AR) also allows users to investigate the scene from their physical viewpoint and spatially relate occluded geometry with the visible world. In case of a disaster scenario, a rescue team can quickly locate an imminent danger, such as a fire or explosion behind an occluder, and proceed with caution.

1.5 Contributions

In this thesis, a framework is presented for instant (i.e., real-time and incremental) *lumigraph* modeling, rendering and *inpainting* of incomplete scenes obtained with an RGB-D sensor. In addition, we present novel interaction techniques powered by image-based rendering to approach the tele-exploration problem as a whole. In the remainder of this section, we explain these contributions in more detail.

Table 1.1: Stages of an image-based rendering pipeline

Stage	Sub-stage	Time unit	Number of views considered
Modeling	View capturing	Per input image	Infinite
	View planning	Per input image	Fixed upper bound
	View refinement	Per novel view	Fixed upper bound
Rendering	View selection	Per stored view	Only views "close" to novel viewpoint
	View blending	Per fragment	Only views with non-zero weight

1.5.1 Real-Time View Planning for Unstructured Lumigraph Modeling

For photometric reconstruction of appearance, we harvest the rich input image stream and organize it in a sensible way. *IBMR* provides methods for using image collections directly in an appearance model. For best results in larger scenes, *IBMR* is usually combined with a dense geometric model, yielding a variant of a lumigraph [41]. Table 1.1 summarizes a typical *IBMR* pipeline, which can roughly be divided into two stages, modeling and rendering.

The objective of the *modeling stage* is to obtain an *IBMR* model. This stage is mostly run offline, in particular, when executed jointly with geometric reconstruction. Conceptually, *IBMR* consists of up to three sub-stages: *View capturing* obtains the input images. *View planning* decides which *views* (images annotated with camera pose) from an input image stream are kept. If a view is kept, *view refinement* tries to rectify any shortcomings of the view data, such as improving the spatial registration or correcting color artifacts.

The *rendering stage* always runs at the target frame rate. Its purpose is to generate, per frame, a novel view for a user-specified viewpoint. Rendering consists of two sub-stages, *view selection* and *view blending*. View selection is concerned with finding appropriate views for sampling, guided by the novel viewpoint. It can operate per frame or per rendered fragment. View blending determines the weights given to individual views for a rendered fragment.

Throughout this pipeline, the number of views considered must be reduced continuously, to make the computational effort of subsequent stages tractable. However, for a basic *IBMR* system, only capturing and blending are mandatory, while the intermediary stages (planning, refinement and selection) are optional. Many systems assume that a reasonable set of views has been captured in advance and concentrate only on the rendering stage, i.e., selection and blending. Another family of techniques attempts to synthesize the best possible view-independent texture map from the available views using offline optimization, concentrating on refinement and selection. In this case, the blending trivially reduces to conventional texture mapping, assuming only diffuse surface reflectance [6, 36, 133]. There are also end-to-end systems, which consider the entire pipeline, but

they typically assume sufficient storage for all desired views. Hence, they have no need for view planning.

We propose an *IBMR* pipeline that can operate in real time, which implies that the input stream must be processed immediately to obtain a photometric reconstruction. Even if we allow a small delay, such as a few seconds or even tens of seconds, for new photometric data to become available, we cannot afford to unconditionally cache all images from an input stream and process them later, since we will never catch up. Instead, we must be able to process them at frame rate. For this purpose, view planning becomes mandatory [120].

We are not the first to consider *IBMR* view planning, but previous work in this area has been limited and not suitable for tele-exploration tasks. Early work [33, 131] only intended to cover each scene primitive (i.e., each triangle or each surface point) once. Some later work considered interactive feedback for view planning [21, 57], but most *IBMR* systems to date only enforce a minimal distance between views and tend to store a large number of redundant views [58]. In contrast, we propose to carefully select optimal views to build an *unstructured lumigraph* [9] that meets the following requirements:

1. Consider coverage of every primitive in every view
2. Operate on an input stream in a strictly incremental fashion [21]
3. Stay within bounded memory

We will address these requirements by incrementally filling a *view store* of bounded size. We will organize the scene using a regular discretization [48]. However, since our plan must be constructed online and in bounded memory, we cannot rely on any global optimization procedures. Instead, for each new view, we propose to derive a *scene coverage score* that anticipates the needs of view selection at runtime to chose the right views during planning.

Our lumigraph view planning is designed to run at interactive rates and deliver quality comparable to using an unbounded number of views. Thus, our work is the first to solve a general view planning problem for *IBMR* at interactive rates and in bounded memory while rendering real-time, as explained in Chapter 3.

1.5.2 InpaintFusion

Inpainting is required for completion of partially scanned scenes and for Diminished Reality (DR), a variant of Mixed Reality (MR) which allows removing objects from the user's perspective view [94] and uncovering otherwise hidden structure in the user's physical environment [66]. Occluded

pixels can be restored from multi-view observations, but not in unobserved areas. Inpainting overcomes this problem by using pixels in the vicinity of a Region of Interest (ROI) and, therefore, does not require additional cameras or pre-recorded observations. If "hallucinated" pixels are acceptable, inpainting has considerable benefits over observation-based methods, in particular, when reconstruction is still ongoing or performed without much preparation.

Conventional inpainting in image space has difficulties to ensure temporal and spatial coherence between frames, such as when rendering stereoscopic images or relighting the background, which requires access to object-space data. For this reason, some inpainting systems heuristically assume an object space by estimating a dominant plane and performing inpainting on a plane embedding. If the dominant plane can be tracked throughout a sequence of frames, the inpainted images can be projected back into the user's view. Such an approach is sufficient for plausible DR, but only if the scene is flat and occlusions can be safely ignored. There are attempts to deform the inpainted result [71], but deformation changes the appearance only in the inpainted plane and never fits the geometry.

In this thesis, we implement a novel approach that inpaints both color and depth. Our method simultaneously searches in the color and depth channel, while minimizing a cost function which combines a color and spatial term in image space with a novel spatial term in object space, which ensures spatiotemporal consistency.

The spatiotemporal inpainting can literally fill in the gaps left by lumigraph modeling:

1. A *DR* mode allows a user to remove or replace a user-specified *ROI* from a view of a scene explored with a live camera. In this case, the live video stream directs the virtual camera, and free choice of viewpoint is not needed; the video stream is used to represent the scene outside of the inpainted area.

2. Exploring areas of the scene that have not yet been observed requires scene completion through inpainting. In this use case, the user employs free-viewpoint navigation, and the inpainting *ROI* is implicitly given by the presence of unexplored locations.

Both modes rely on the same inpainting framework, although they address the needs of different use cases and applications. The inpainting framework will directly re-use the *IBMR* framework, so that either inpainting mode can be combined with view-dependent texture mapping and free-viewpoint navigation. More details of this technique are explained in Chapter 4.

1.5.3 Drone-Augmented Human Vision

As mentioned in Section 1.5.3, *IBMR* can greatly contribute to the tele-exploration task, even when interactions are needed. In order to show this impact, in this thesis, we demonstrate the first proof-of-concept implementation of *drone-augmented human vision*. A system to give new abilities to users with *IBMR* powered interactions and visualization.

In particular, we focus on drones because of their proven contributions to tele-exploration in recent years. In particular, their high degree of maneuvering capabilities makes them ideal robots for exploring hard to reach areas.

We couple an indoor drone with a *HMD* to deliver an exocentric perspective on the drone, letting the pilot control the drone via gaze direction. The drone carries an autopilot, but relies on external tracking, since we wanted optimal flight stability for our prototype. We present a first experiment showing how virtual exocentric visualization supports spatial understanding and thus enables exploration and natural interaction with the drone. In a second experiment, we use *VR* for its virtual viewpoint nature and compare it with the physical viewpoint that is additionally provided by *AR*.

VR provides a synthetic exocentric view by combining the live video with a 3D model of the occluded environment from any viewpoint, independent of a user's physical viewpoint. Using image-based rendering of the drone's video stream delivers a realistic impression with live updates [99]. By coupling the drone autopilot to the user's gaze direction, the experience is redefined from remotely piloting a drone to perceiving the occluded world with *drone-augmented human vision*.

As a special case of *VR*, *AR* additionally adds the illusion of X-ray vision: A pilot wearing a see-through display can make the walls or other occluders partially transparent to reveal the area currently observed by the drone. More detail of this technique is explained in Chapter 5.

1.6 Collaboration Statement

The contributions mentioned in this thesis have been peer-reviewed and published as part of the papers listed below. We provide information on the collaborations which occurred during the work on these publications. For all papers, the author contributed to the development work, which involves design, implementation and analysis, as well as to the publication work. In particular:

- Okan Erat, Alexander Isop, Denis Kalkofen and Dieter Schmalstieg, "Drone-Augmented Human Vision: Exocentric Control for Drones Exploring Hidden Areas," *in IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 4, pp. 1437-1446, April 2018

The author created user interactions and implemented image-based rendering in an *HMD*. In addition, the author prepared the user study. Dr. Werner Alexander Isop assembled a lightweight indoor drone, created APIs to communicate with it and he helped setting up the enviroment for the user study. Prof. Dr. Dieter Schmalstieg and Dr. Denis Kalkofen provided guidance with development, user study, writing the publication and introduced ideas through out the project.

- Okan Erat, Markus Hoell, Karl Haubenwallner, Christian Pirchheim and Dieter Schmalstieg, "Real-Time View Planning for Unstructured Lumigraph Modeling," *in IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 11, pp. 3063-3072, Nov. 2019

  The author implemented the per-voxel view planning algorithm and keyframe pose and intensity optimization. Markus Hoell contributed to the shader programming and Unity 3D development. He also presented valuable ideas. Karl Haubenwallner supported the project by implementing a testbed enviroment and Hilbert curve generator. Dr. Christian Pirchheim helped with the dataset processing. Prof. Dr. Dieter Schmalstieg provided guidance on methods, development, ideas and publication.

- Shohei Mori, Okan Erat, Wolfgang Broll, Hideo Saito, Dieter Schmalstieg, Denis Kalkofen "InpaintFusion: Incremental RGB-D Inpainting for 3D Scenes." *IEEE Transactions on Visualization and Computer Graphics* 26.10 (2020): 2994-3007.

  The author and Dr. Shohei Mori both contributed to development and publication, while the other authors contributed in varying degrees. In Particular: The author implemented a pipeline to render 3D skeletal animation and to have physics interactions with the inpainted point cloud. In addition, the author created a data set using new depth cameras and implemented an experimental keyframe selection pipeline. Most notably, the author combined *IBMR* system with *DR* for scene completion results. Dr. Mori implemented the core of the inpainting engine. Prof. Dr. Wolfgang Broll, Prof. Dr. Hideo Saito, Prof. Dr. Dieter Schmalstieg and Dr. Denis Kalkofen provided guidance on methods, development, ideas and publication.

2 Related Work

Contents

In this chapter, we first provide a context to our research work by summarizing the state of the art of tele-exploration. Later, in the following subsections, we analyze related work separated by subject area, in regards to tele-exploration.

In order to visually inspect a remote environment in real-time, a minimum requirement is to have at least one sensor at the remote location that can stream images or sensory information to reconstruct and render the environment. For the sake of the exploration task, the sensor has to be mobile. According to these requirements, we aggregated related work for tele-exploration under the following two headings:

Image streaming from teleoperated robots. According to Lichiardopol [85], teleoperated robots can be grouped according to their application as: *Military/Defense robots* [8], *Security robots* [20, 51], *Underwater robots* [87] and *Telesurgery robots*. One common problem with these systems is the lack of flexibility concerning the viewpoint. A remote user's ability to change the viewpoint at the remote location is restricted by the degrees of freedom of the robot and the network speed. In order to overcome this problem, scene reconstruction techniques were utilized.

Scene reconstruction. Structure-from-motion algorithms can provide a sparse [74] or dense point cloud [29] or a 3D mesh [97]. Later this 3D information with the captured per-point colors can be

used to render the scene from any viewpoint. Recently, collaborative approaches using multiple drones soared in popularity. Schmuck et al. constructed individual Simultaneous Localization and Mapping (SLAM) maps of the environment at each drone and later unified them in the server [118]. There also exists research to construct environment maps using multiple handheld devices [34, 95, 102, 129]. However per-point coloring is not intuitive enough for the tele-exploration scenario given that remote users want to have feeling of telepresence. Also, despite providing a flexible viewpoint, these techniques do not fully take advantage of a reconstructed enviroment, which could be used to interact with the remote scene from user's own viewpoint.

2.1 Image-Based Rendering

Reconstructed models can be organized according to the amount of geometric and photometric data they contain. Pure geometric models do not contain any photometric information, while pure lightfields [82] do not contain any geometry. Popular real-time reconstruction methods, such as volumetric fusion from RGB-D sensors [101], deliver detailed geometry (e.g., as a Truncated Signed Distance Function (TSDF)), but only minimal photometric data, usually consisting of averaged colors, either per voxel or, if a mesh is extracted, per surface vertex. Averaging often leads to a loss of contrast and a blurry appearance, even if colors are cached densely on the surface [26].

Offline reconstruction. Offline methods which assume that detailed geometry has already been recovered concentrate on extracting an optimal texture map from a set of input images (views) [6, 36, 133, 141]. However, even with perfect registration of views to geometry, baking the image information from multiple views into a single texture map destroys view-dependent aspects.

Image-based rendering. Image-Based Modeling and Rendering (IBMR) gives a stronger emphasis to photometric information by subsampling a view-dependent plenoptic function [12]. For small objects or scenes, pure *IBMR* methods rely exclusively on densely sampled views, while replacing detailed scene geometry with crude proxy geometry (e.g., a single plane or a sphere). For such *outside-in* scenes, the possible viewpoints are typically restricted to an orbit around the object or even a narrow zone inside such an orbit.

Pure *IBMR* would have excessive storage requirements for larger scenes, where free-viewpoint navigation is most desirable. For such *inside-out* scenes, better storage efficiency can be obtained by combining more detailed geometry with sparser views.

Geometry-based *IBMR*, such as the lumigraph [41], combines a (semi-)dense proxy geometry

with sampled views. With view-dependent texture mapping [24], we can an build an unstructured lumigraph [9] and render new views directly from sparse, irregular views. However, poorly registered views may lead to blurriness and ghosting artifacts.

A lot of research has addressed better registration, e.g., by using floating textures [28] or resampling views into a *surface lightfield* [15] indexed per surface point instead of per viewpoint. With surface lightfields, one can either render view-dependent appearance convincingly [89, 104], or attempt inverse rendering [64, 111]. Intrinsic image decomposition is even possible in real time for small scenes [43, 91], allowing interactive relighting [90]. If real time is not needed, reconstructions can be automatically corrected and completed [59] or even redesigned [140].

Forward-projection *IBMR* is an alternative to geometry-based *IBMR* which creates distinct geometry per input view, for example, from superpixels [11, 13] or local structure from motion [48]. These methods can better suppress artifacts caused by sparse or incomplete geometry.

View planning. Besides registration, a second challenge for geometry-based *IBMR* is that it does not trivially scale to a large number of views. This implies that the view planning deserves specific attention in *IBMR*. If a finite number of targets views is known in advance, view planning can aim to optimally cover these views [54, 142]. If the target views can be assumed to lie within a bounded "walking" range, view planning can be reduced to the *art gallery problem*, aiming to cover every surface point in the scene with at least one view [33, 131]. Unfortunately, texturing with a single view is not sufficient for high quality *IBMR*. It is more meaningful for modern *IBMR* to consider a reprojection error between sampled views, to decide if additional views should be acquired [21].

For instance, recent work [48] has proposed offline view planning for larger scenes by subdividing the scene into parts and establishing an explicit mapping between views and scene parts. However, this work does not place an upper bound on the size of the view cache, and does not run in real time. This makes it unsuitable for interactive reconstruction [21] or telepresence [27, 37, 103].

View planning for outside-in scenes has been demonstrated at interactive rates to guide a user's acquisition with a handheld camera [21, 57]. In robotic surveying, there exists the related problem of "next best view planning".

Our view planning has the additional requirement that it needs to consider surrounding geometry of an inside-out scene and not only the orbital space around an outside-in scene. We show how lumigraph modeling can be cast as a local, incremental problem, which can be solved *online*. Hence, our method fills a *texture store* of configurable size with the best views. The resulting model can be instantly used for free-viewpoint rendering.

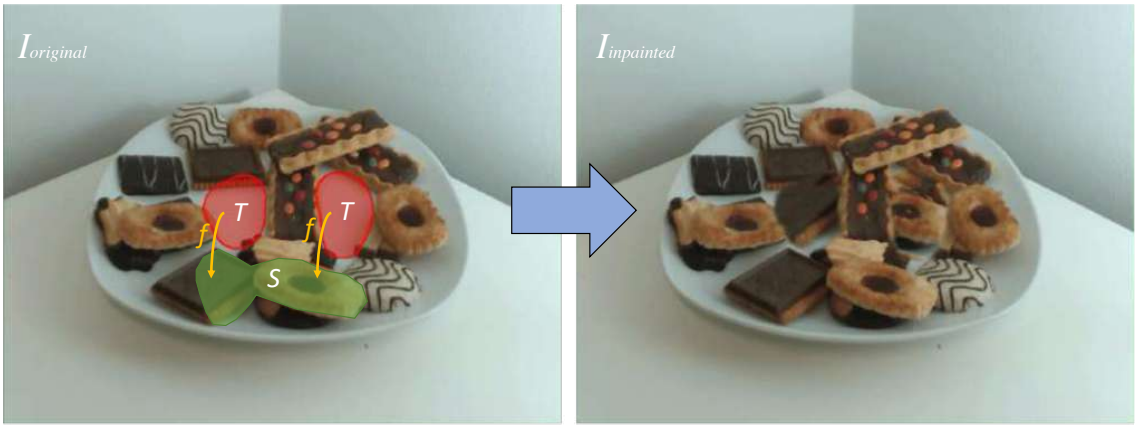


Figure 2.1: Illustration of the notation used in inpainting. The transformation function f maps target region T being inpainted to source pixels S.

Table 2.1: Qualitative comparison of inpainting/Diminished Reality (DR) literature.

Literature	Scene	Object detection	Object tracking	AR Depth
Siltanen [122]	Plane	Marker region	*6DoF* (Marker)	No
Korkalo et al. [77]	Plane	Marker region	*6DoF* (Marker)	No
Herling and Broll [49]	Plane	Interactive drawing (one-view)	3DoF (Contours)	No
Herling and Broll [50]	Plane	Interactive drawing (one-view)	3DoF (Contours)	No
Kawai et al. [72]	Planes	User drawing (multi-views)	*6DoF* (*SLAM*)	No
Siltanen [123]	Planes	User drawing (one-view)	*6DoF* (*SLAM*)	No
Kawai et al. [71]	Curved plane	Marker region	*6DoF* (Marker)	No
Proposed method	3D scene	Interactive drawing (one-view)	*6DoF* (*SLAM*)	Yes

2.2 Inpainting

The simplest form of DR, using only a single image, replaces pixels in a target region T, $T \in I$, of an image I with pixels from sources $S \in I$. Therefore, we need to find the transformation $f : T \rightarrow S$ that preserves consistency in the appearance between the target region T and the remaining image $\overline{T} = I \setminus T$ (see Figure 2.1). Furthermore, DR methods need to support motion in 3D space with Six Degree of Freedom (6DoF). This implies that, after inpainting, T and $\overline{T}$ need to be consistent under arbitrary motion of the camera. Previous DR methods mainly differ in how the function f is defined and which sources S are considered. Therefore, in this section, we review previous approaches (see Table 2.1).

2.2.1 Multi-View Approaches

One direction of research has focused on rendering occluded pixels from multiple different live video observations of the hidden area, while another direction first captures the scene from multiple camera positions using a single camera. An early example of the former case is the multi-view paraperspective projection model proposed by Zokai et al. [143] that uses an additional calibrated cameras as S to search for background patches in T with a similar appearance. Meerits and Saito [88] use additional RGB-D frames from a Microsoft Kinect sensor as S to observe the background with depth information. The work of Cosco et al. [18] creates DR from multiple images that have been captured over time. They propose a system recording images as S, before the object to be diminished is placed in the scene. While Cosco et al. use the multi-view data immediately after capturing, Li et al. [84] use older images from Internet photo collections, registered in 3D space as S.

The above methods assume calibrated multi-view cameras to define the mapping f under epipolar constraint [46]. This enables a fast pixel search in S at the price of relying on dense observations, which may have to be generated in advance or at runtime using additional cameras. Either restriction makes these approaches difficult to apply to mobile applications.

2.2.2 Video Inpainting

A more flexible approach for DR is inpainting, which can be defined as the global optimization of the transformation function $f : T \rightarrow S$ in which $S \equiv \overline{T}$, i.e., $f : T \rightarrow \overline{T}$ [50].

Inpainting for DR originates from research on video restoration. The primary difference between image and video inpainting is that video inpainting makes use of the pre-recorded image sequence as an inpainting source S [61], instead of just a single frame. Thus, it can be defined as a global optimization problem of finding the best transformation function $f : T \rightarrow S$ where $S \equiv \overline{T}_i$ at frame F_i.

Wang et al. [134] presented pioneering work in this area. They separate the pre-recorded scene into several layers using dominant optical flow, and showed that rendering all layers except one results in a scene without the selected object. Lepetit et al. [81] take pixels from $\overline{T}_j$ in frame $F_j, i \neq j$, to inpaint T_i by reprojection via a reconstructed background triangle mesh. Shen et al. [121] find a linearly moving foreground object in a geometrically aligned temporal texture space and propagate non-occluded pixels in $\overline{T}_j$ to the occluded pixels in T_i. Klose et al. [76] use a point cloud for inpainting, where point reconstructions from $\overline{T}_j$ are sampled through pre-defined filters to fill in T_i.

Although video inpainting methods generate plausible results, they cannot be used in DR applications, since they rely on costly global optimizations and consider both past and future frames. DR must be able to react instantaneously to changes in the user's viewpoint, using only past information, while maintaining coherent visual appearance over time. Furthermore, usage of past frames is typically limited to the previous frame or a small number of frames.

2.2.3 Image and Depth Inpainting

In contrast to video inpainting, image inpainting takes pixel information only from a single image. The most popular image inpainting approaches are based on the idea of searching for patterns in the image which are similar to a region placed over the boundary of T and $\overline{T}$ [19]. The creators of the PatchMatch method [3, 4] report on two key insights for finding a near optimal f: They use randomized searching for corresponding patches in $\overline{T}$, and they make use of propagation of the searched offsets to the adjacent pixels in T. These two insights enable generating consistent reconstructions.

However, PatchMatch is not designed for predictable real-time performance, and does not consider temporal coherence over image sequences. Therefore, Herling and Broll [49] proposed PixMix, a method relying on frame-to-frame propagation of patches to accelerate the search and ensure temporal coherence. Later, they improved image quality and runtime of their method [50] by applying a homography transformation (estimated between an earlier keyframe and the current frame) to the reference map $f : T \rightarrow \overline{T}'$, where $\overline{T}'$ represents T transformed by the homography. Kunert et al. [78] extended the method by combining it with observed background pixels. Kawai et al. [72] and Siltanen [123] also extended this strategy to enable processing of several planes in parallel. Kawai et al. [71] furthermore proposed an inpainting algorithm that deforms inpainted results using feature point tracking.

All these attempts assume that the scene is locally planar. While this notion makes it easier to obtain real-time performance, it cannot recover depth information in T. Advanced AR rendering typically requires the evaluation of lighting, occlusion and other view-dependent phenomena [23, 112], as well as image synthesis for stereoscopic displays [31]. Without restoring proper depth information in the inpainted area, such rendering methods cannot be properly supported.

Our work is also conceptually related to depth densification. Unlike offline structure-from-motion methods, AR requires densification to operate in real-time. State-of-the-art methods densify sparse *SLAM* maps [56] or perform real-time short-baseline stereo matching [130]. We could use such methods as alternative forms of reconstruction, but, of course, they cannot deal with unobserved

Figure 2.2: Mirk and Hlavacs created a virtual tourist application providing a stereo view.

areas.

Recently, Generative Adversarial Network (GAN) methods [40], have shown great promise for complex image synthesis [60, 105, 138, 139]. Inpainting based on *GAN* essentially uses a database of trained feature as S. Such approaches have also been shown to be able to generate depth maps from color images [38, 80] or inpaint RGB-D images [25]. However, *GAN* typically requires images to be resized before feeding them into the network, and, again, on the output side. These implicit resampling steps make the results prone to aliasing problems, when the inpainted area changes with perspective distortion (another resampling step), as the camera pose changes from frame to frame. In contrast, our approach inpaints color and depth using a conventional patch representation, which does not have to be scaled or resampled. It also has the advantage that it works instantaneously without requiring extensive training databases to be collected and processed.

2.3 Drone Control

Existing work on occluded or remote space discovery with drones proposes a variety of interaction techniques to steer the drone and visualize the data coming from its sensors. Depending on the visualization of the sensor data, mostly from cameras, related work can be categorized into egocentric control and exocentric control. Moreover, our work is related to remote visualization techniques involving live video.

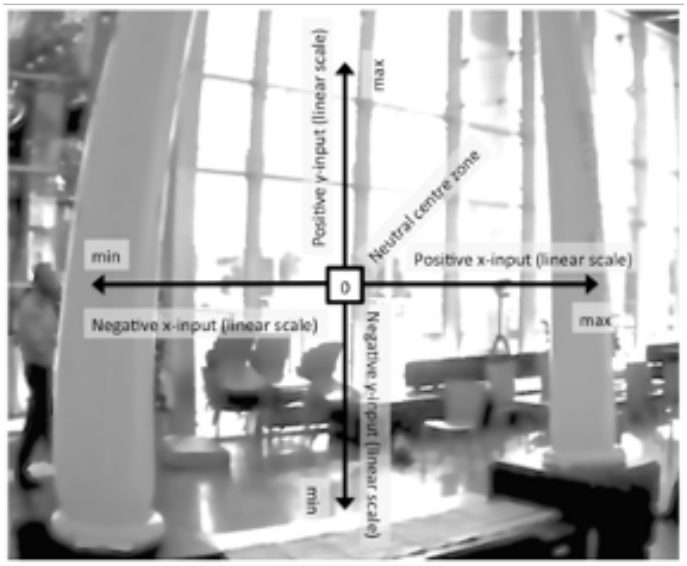


Figure 2.3: Hansen et al. let users control a drone by gazing at the camera view streamed from the drone.

2.3.1 Egocentric Drone Control

Egocentric control techniques visualize camera images from the first-person view of the drone and immerse the user into the remote location currently occupied by the drone. Using a Head-Mounted Display (HMD) to display video from a drone, Mirk and Hlavacs [93] created a virtual tourist application (see Figure 2.2). However, the user was not given full control of the drone to prevent crashes; only the user's head movements were translated into the yaw rotation of the drone. Hansen et al. [44] capture eye gaze, while the drone pilot is looking at the camera stream (see Figure 2.3). The 2D vector formed between the screen center and the point gazed at on the screen is mapped to speed and rotation around a 3D axis in the drone's local frame. As humans tend to rapidly change their gaze direction, this technique may be problematic for flight control whenever the pilot looses concentration.

Higuchi et al. [52] synchronize head movements of the user with a drone, except for pitch and roll rotations (see Figure 2.4). While this gives an intuitive control, the latency between the pilot's movements and response of the drone can quickly create motion sickness. In addition, the motion dynamics of the drone make it impossible for the drone to exactly replicate the path taken by the pilot's head, negatively affecting the spatial understanding of the human. As summarized by Chen et al. [14], egocentric robot control presents the user with several problems, the most severe ones being narrow Field of View (FOV), orientation and altitude misjudgement, and a general lack of scene understanding.

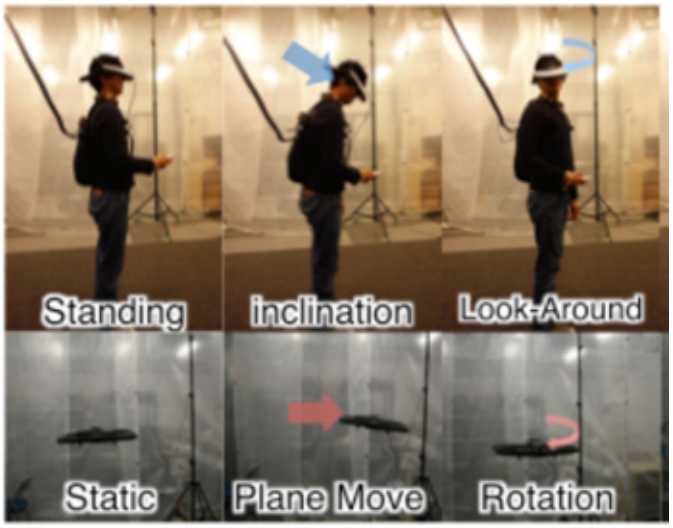


Figure 2.4: Higuchi et al. synchronize head movements of the user with a drone.

Figure 2.5: Kashara et al. allow users to control the drone with a touch screen device in their own reference frame and map control commands into the drone's local coordinates.

2.3.2 Exocentric Drone Control

In contrast to egocentric control, an exocentric control technique steers the drone while the user is observing it directly. As discussed by Cho et al. [16], exocentric drone control is prone to accidents due to left-right confusion between user's and drone's local coordinate frames. Kashara et al. [69] tackle this problem by allowing users to control the drone with a touch screen device in their own

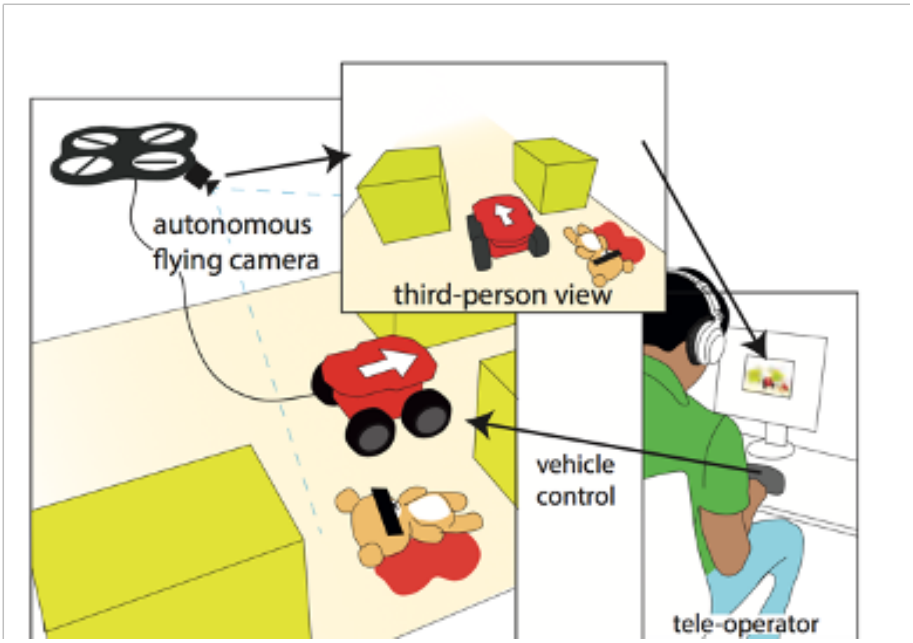


Figure 2.6: Saakes et al. use drone camera to observe a ground robot from a third person view and control it.

reference frame and mapping control commands into the drone's local coordinates automatically. However, the users had to observe the drone with the device's camera for pose estimation and move it on the 2D screen, which is not possible in the presence of occlusions. In addition, 2D gestures do not allow for an intuitive interface for generating a motion vector that is a combination of axes. Similar to Kashara et al., Hashimoto et al. [47] also provides a touch screen based control, but they place a camera at a fixed viewpoint to observe the robot (see Figure 2.5). This is not feasible during an investigation of an occluded scene. Saakes et al. [114] uses a drone camera to observe a ground robot from a third-person view (see Figure 2.6). In an unknown occluded environment, using another robot just increases the complexity. Sugimoto et al. and Hing et al. [53, 126] provide a visualization to observe the robot from an exocentric point of view. However, their system limits the freedom of the viewpoint and makes it hard to relate surrounding colliders to the robot. Karanam et al. [68] use WiFi signals transmitted by drones to monitor them behind the occluding structures.

Zollmann et al. [144] focuses on the spatial understanding problems that arise when the drone is far away from the user (see Figure 2.7). They use an exocentric Augmented Reality (AR) display based on the backfacing camera of a handheld tablet. The drone's altitude over the terrain and distance to the user is visualized in 3D on top of the video. However, if the drone faces dense obstacles in close proximity, this technique does not provide a detailed enough visualization for accurate control. Bergé et.al. [5] create a synthetic point cloud resembling a 3D reconstruction

Figure 2.7: Zollmann et al. visualize drone's altitute over the terrain and distance to the user in 3D.

obtained by a drone and visualize it in immersive VR. They also develop a method to evaluate the difficulty of finding a target.

All these techniques demonstrate the potential of using an exocentric viewpoint for drone control, but do not allow for easy and intuitive navigation. Introducing direct manipulation for this purpose is the main contribution of our work.

2.3.3 Visualization of Remote or Occluded Information

Simulating X-ray vision for the purpose of revealing hidden infrastructure has been a goal of *AR* research for a long time [32]. Most of the X-ray vision techniques compose a video image with purely virtual information or simulate a cutaway of the occluder [17].

Our work also relates to research on visualizing and interacting in a distant environments and, by extension, to telepresence systems. For example, remote visualization at real-world scale was presented by Kasahara et al. [70]. The system provides omnidirectional remote visualization, enabling a user to participate in the remote user's application. While the system allows to decouple orientation, it does not provide control over the user's position.

Neumann et al. [99] proposed the idea of surveillance based on augmented video environments, which rely on projective texture mapping of live video to a reconstruction of an outdoor environment. For indoor surveillance, presenting the video streams in the context of a spatial model rather than via a more conventional multi-windowing display was explored by Wang et al. [135].

These systems assume an observer in a control room, but similar ideas have been explored for mobile users. Kameda et al. [67] report on a mobile *AR* system displaying registered video streams from remote cameras. Avery and Sandor [2] use ghosted-view X-ray vision to look through walls. Their system shows videos received from a remote robot controlled by the user via joystick [1].

Sandor et al. [116] later proposed a “melting” metaphor to disocclude buildings. Sandor et al. [115] show a method for X-ray rendering using salient features of occluders.

Another aspect of remote information display is camera navigation. For example, Mulloni et al. [96] describe how to transition between the video from multiple cameras placed in an outdoor environment without loosing spatial context. Hoang et al. [55] investigate remote viewpoint manipulation for close-up observation. We draw inspiration from all of these methods, but additionally control the flight path of a drone indirectly by introducing interaction techniques for the interactive definition of the desired viewpoint.

3 View Planning and Rendering

Contents

In this chapter we introduce a novel Image-Based Modeling and Rendering (IBMR) pipeline to answer the requirements of tele-exploration applications. We are not the first to design an *IBMR* system, however, we adapt *IBMR* for the online nature of the problem which requires real-time performance and memory management for unknown scene sizes (see Figure 3.1 and 3.2).

We voxelize a scene and make sure to keep images that best cover parts of the scene with different viewing angles and distances from the geometry. When considering a new image we aim for keeping images that have as unique information as possible for the scene, while removing images that are less valuable for the scene.

We additionally minimize pose errors related to the Simultaneous Localization and Mapping (SLAM) system before registering any image into our system. Images are further optimized for their intensity consistency.

3.1 Method

For building an *IBMR* pipeline, we take advantage of the inherent multi-threading of *SLAM* [75]. *SLAM* typically uses a tracking task running at full frame-rate and a slower mapping (i.e., geometric

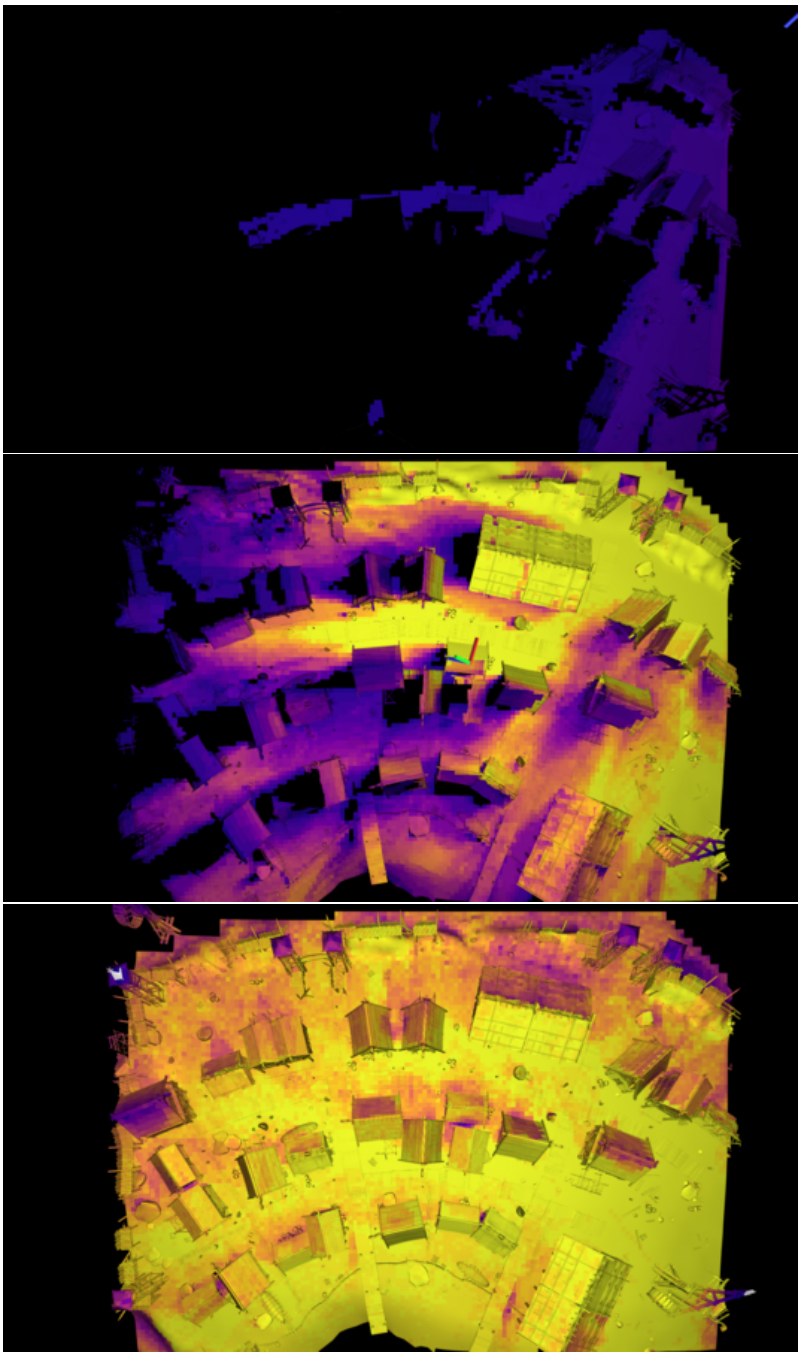


Figure 3.1: Progression of real-time view planning: unseen parts of the scene are black, and brighter colors (purple to yellow) mean more views covering a portion of the scene. Gray color indicates the surface has been seen but could not be textured.

reconstruction) task. We introduce additional tasks for photometric registration and novel view

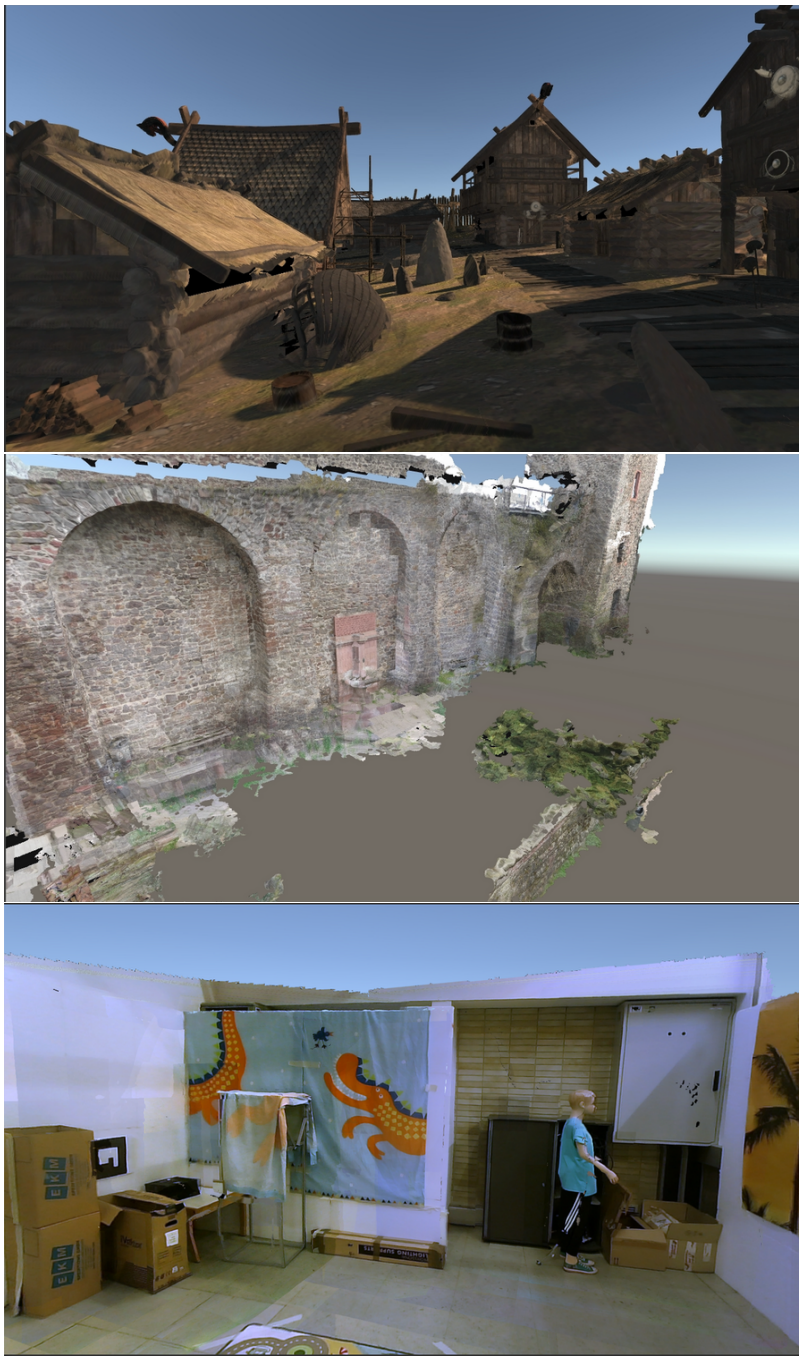


Figure 3.2: Images rendered using the unstructured lumigraph.

synthesis, which run at their own, independent rates.

Figure 3.3 shows an overview of our system architecture: White boxes belong to the existing *SLAM* system, which fills the geometry store. Red boxes describe our novel photometric registration.

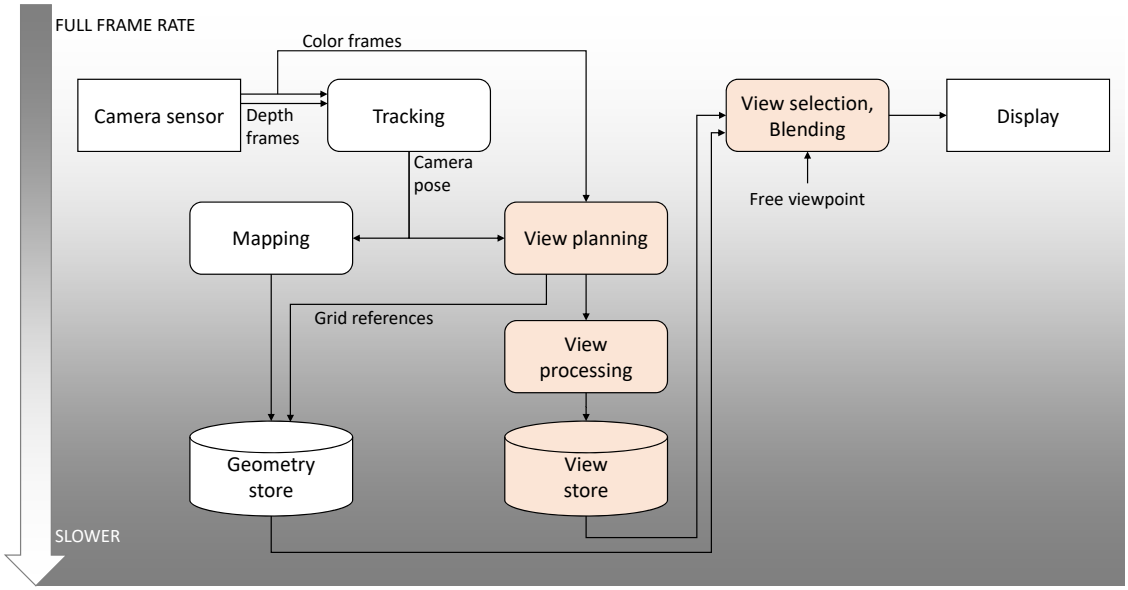


Figure 3.3: We extend *SLAM* with lumigraph modeling and rendering including a novel real-time view planning approach. This diagram shows existing components in white and new components in orange.

Every input frame is considered by the view planning component as a potential novel view to be placed in the view store, which is also associating novel views with the scene geometry.

A lumigraph rendering component is responsible for generating novel views. Using the information from the geometry and view stores, this component performs view selection and blending to generate a novel view to display. The framerate of the rendering component is decoupled from the lumigraph modeling, and rendering can even run remotely in a telepresence environment.

We begin by reviewing a basic unstructured lumigraph (Section 3.1.1), as described by Buehler et al. [9]. Then, we explain our core contribution, the real-time view planning algorithm for the lumigraph (Section 3.1.2). We also discuss how the views selected for the lumigraph can be refined in real time (Section 3.1.3), and, finally, how the runtime view selection works (Section 3.1.4).

3.1.1 Basic Lumigraph Blending

Basic lumigraph blending creates novel views by sampling views from a *view store* $F = \{F_i\}$. Each view $F_i = (\mathbf{M}_i, C_i, \mathcal{D}_i)$ consists of a camera pose $\mathbf{M}_i$, decomposed into camera position $p(\mathbf{M}_i)$ and viewing direction $\hat{d}(\mathbf{M}_i)$, as well as a color image $C_i(\mathbf{u})$ and a linear depth image $\mathcal{D}_i(\mathbf{u})$.

Lumigraph blending selects, for each sample position $\mathbf{p}$, the best n views (collected in F_n) according to a weighting function w_i, and blends them into a new view $F_o = (\mathbf{M}_o, C_o, \mathcal{D}_o)$. Using a projection

operator $\pi(\mathbf{p}, F_i)$ from world space to image space and an inverse projection operator $\pi'(\mathbf{u}, F_o)$ from image space to world space, we obtain C_o as follows:

$$C_o(\mathbf{u}) = \sum_{F_i \in F_n} C_i(\pi(\pi'(\mathbf{u}, F_o), F_i)) \cdot \frac{w_i(\mathbf{u})}{\sum_{r \in F_n} w_r(\mathbf{u})} \tag{3.1}$$

The weights w_i are obtained by combining terms that describe the geometric (i.e., directional and positional) similarity of a reference view to the novel view. The directional term $w_i^{\hat{d}}$ is described using a clamped cosine of the angle between the reference view and the novel view.

$$w_i^{\hat{d}}(\mathbf{u}) = \max(0, \frac{p(\mathbf{M}_o) - \pi'(\mathbf{u}, F_o)}{\|p(\mathbf{M}_o) - \pi'(\mathbf{u}, F_o)\|} \cdot \frac{p(\mathbf{M}_i) - \pi'(\mathbf{u}, F_o)}{\|p(\mathbf{M}_i) - \pi'(\mathbf{u}, F_o)\|}) \tag{3.2}$$

The position term w_i^p is described as the ratio of distances to the camera center:

$$w_i^p(\mathbf{u}) = \max\left(0, 1 - \frac{|p(\mathbf{M}_i) - \pi'(\mathbf{u}, F_o)|}{|p(\mathbf{M}_o) - \pi'(\mathbf{u}, F_o)|}\right) \tag{3.3}$$

In addition to the geometric similarity, the weight must also ensure that a sample is valid. Buehler et al. [9] only consider that a sample $\mathbf{p}$ must be within the field of view covered by F_i (assumed to have an opening angle of 2α). This suffices if the geometric model is very simple, but, for complex geometric models, we must additionally take care that a sample $\mathbf{p}$ is not occluded in F_i and that $\mathbf{p}$ is not closer than Δ_{xy} to a depth discontinuity larger than Δ_z [103], which would make $\mathbf{p}$ unreliable. We combine these constraints in a validity function $v(\mathbf{p}, F_i)$ as follows:

$$v(\mathbf{p}, F_i) = \begin{cases} 0, \text{ if } |\mathbf{p} - p(\mathbf{M}_i)| > \mathcal{D}_i(\pi(\mathbf{p}, F_i)) \\ 0, \text{ if } \frac{\mathbf{p} - p(\mathbf{M}_i)}{\|\mathbf{p} - p(\mathbf{M}_i)\|} \cdot \hat{d}(\mathbf{M}_i) < \cos\alpha \\ 0, \text{ if } \max\limits_{\Delta_{xy} \in \{\pm 2\}^2} |\mathcal{D}_i(\pi(\mathbf{p}, F_i)) - \mathcal{D}_i(\pi(\mathbf{p}, F_i) + \Delta_{xy})| > \Delta_z \\ 1, \text{ otherwise} \end{cases} \tag{3.4}$$

The final weight is obtained by linearly combining the geometric similarity and multiplying by the validity function using a parameter λ:

$$w_i(\mathbf{u}) = v(\mathbf{u}, F_i) \cdot (\lambda \cdot w_i^{\hat{d}} + (1 - \lambda) w_i^p) \tag{3.5}$$

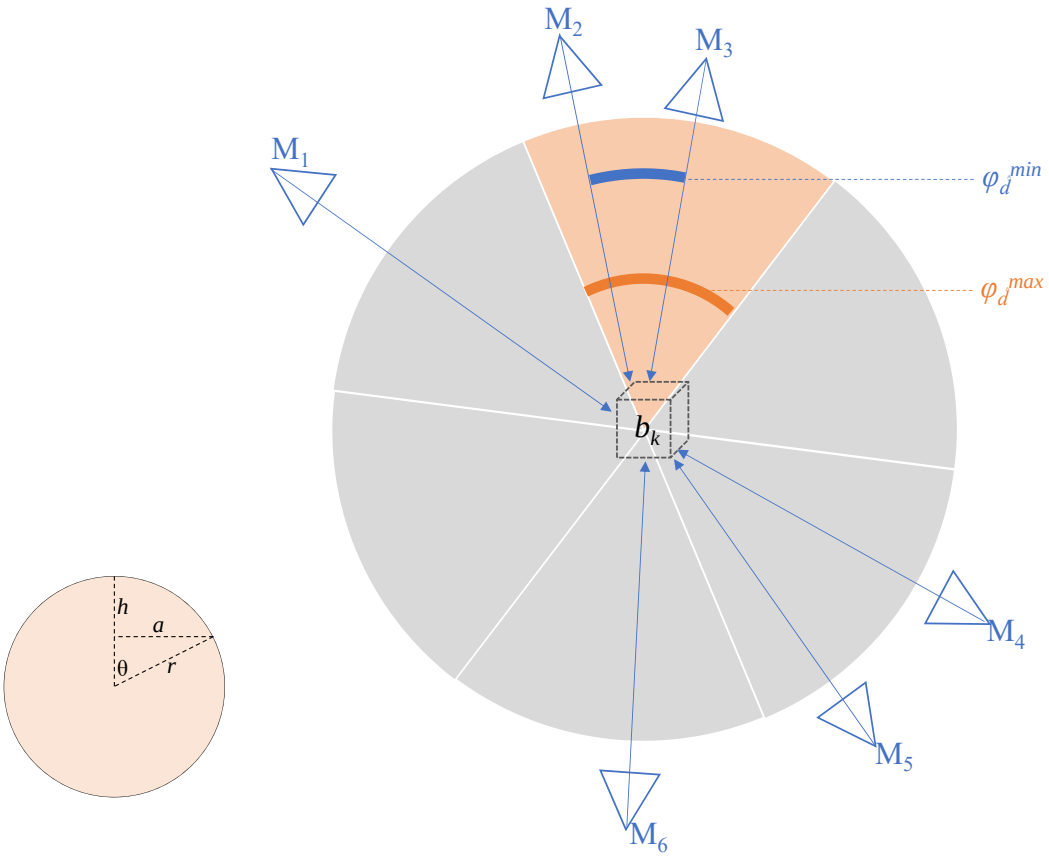


Figure 3.4: (left) By dividing the area of a unit sphere, 4π, by the number of views, we obtain a maximum area per view. The area assigned to one view has as an upper bound of the area of a sphere cap, $2\pi(1-\cos\theta)$. (right) Example for angular coverage of b_k observed by six views $c_1,\ldots,c_6$. The angular coverage weight is related to the ratio of the minimal observed angle between two cameras, $\varphi_{\hat{d}}^{min}$, and the maximum possible angle between two cameras, $\varphi_{\hat{d}}^{max}$.

3.1.2 View Planning

The above description of basic lumigraph blending assumes that F is small enough so all views can be stored and searched at runtime. The key contribution of our paper is the introduction of a real-time solution for *view planning*, which addresses two requirements not handled by basic lumigraph blending: first, choose views from the incoming image stream to store in F, second, obtain a pre-selection so view blending can be done with a constant effort that is independent of the chosen size of F.

Our view planning approach extends the frame store used in lumigraph blending with an additional view-independent data structure: We organize the scene into a regular grid $B = \{b_k\}$, which sub-

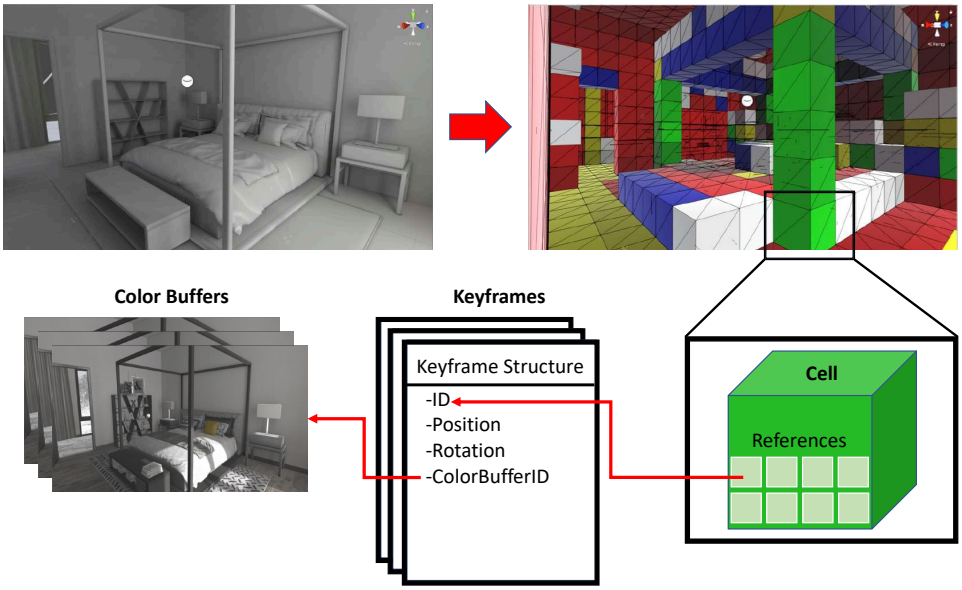


Figure 3.5: Illustration of cell grid representation of an example scene geometry. One single cell is chosen for demonstrating keyframe store F referencing from cells. Cells are randomly colored for visual contrast in the illustration.

divides the scene geometry G into cells $G(b_k)$. Per cell b_k, we store a set of R^{max} references to F, denoted as $R(b_k) = \{F_i^{k,r}\}, 1 \leq r \leq R^{max}$ (see Figure 3.5).

The cell structure has a number of advantages: It reduces the overall effort compared to processing surface geometry explicitly, it exploits spatial locality, and it decouples the lumigraph from the detailed surface geometry reconstruction. Only during the final rendering are the views associated with individual surface points through indirect texture lookups. The geometric and photometric reconstructions can evolve independently, making our approach robust to variations in computational load and other unforeseen challenges that may occur in a real-time system. For example, new views can be incorporated to refresh the view store after changes to scene geometry or incident illumination. Moreover, cells naturally correspond to blocks of a sparse volumetric data structure, which is now commonly used for large scenes [65].

Coverage metric In order to fill the view store with the best views, we define a *coverage* metric that expresses the benefit of a new view in covering the lumigraph. We weight two quality criteria, each expressed by a factor in the range [0,1], which can be seen as a view-independent variant of the directional and positional similarity described above:

- **Directional coverage** $\varphi_{\hat{d}}$: Views observing a cell should be well-distributed in the cell's orientation space, so that every new view covers a portion of the scene from a new angle.
- **Positional coverage** φ_p: Views should have the desired pixel density (neither too dense nor too sparse). Moreover, the view should see as much as possible of the surface inside the cell.

We compute the overall coverage φ for a frame F_i as a weighted sum of directional and positional coverage:

$$\varphi(F_i, b_k) = \lambda \cdot \varphi_{\hat{d}}(F_i, b_k) + (1 - \lambda) \cdot \varphi_p(F_i, b_k) \tag{3.6}$$

Directional coverage. We determine the minimum angular deviation $\varphi_{\hat{d}}^{min}$ to all views F_j already selected for a particular cell (represented by its centroid, $p(b_k)$):

$$\gamma = \max_{F_j \in R(b_k)} \frac{p(\mathbf{M}_i) - p(b_k)}{\|p(\mathbf{M}_i) - p(b_k)\|} \cdot \frac{p(\mathbf{M}_j) - p(b_k)}{\left\|p(\mathbf{M}_j) - p(b_k)\right\|} \tag{3.7}$$

$$\varphi_{\hat{d}}^{min}(F_i, b_k) = \delta(|R(b_k)|, 0) + (1 - \delta(|R(b_k)|, 0)) \cdot \gamma \tag{3.8}$$

Here, δ denotes the Kronecker delta function. If more views are stored in $R(b_k)$, the angle between them must become smaller. For R^{max} views (all references are filled), the Tammes number denotes the maximum angle $\theta(R^{max})$ between views [79]. We can estimate an upper bound (Figure 3.4, left) to the cosine of the Tammes number as $\varphi_{\hat{d}}^{max} = \cos(\theta(R^{max})) = 2 \cdot (1 - 2/R^{max})^2 - 1$. Therefore, we obtain an approximate angular coverage $\varphi_{\hat{d}}$ from the minimal observed angle between views and the largest possible angle between views (Figure 3.4):

$$\varphi_{\hat{d}}(F_i, b_k) = \min\left(1, \frac{1 - \varphi_{\hat{d}}^{min}(F_i, b_k)}{1 - \varphi_{\hat{d}}^{max}}\right) \tag{3.9}$$

Positional coverage. Positional coverage combines a term judging the resolution of I with respect to b_k and a term describing what fraction of b_k is visible. The distance of a cell to the view is given by d:

$$d(F_i, b_k) = |p(\mathbf{M}_i) - p(b_k)| \tag{3.10}$$

We express how well the distance of a view matches an ideal distance d_{max} using a Gaussian g with variance σ^2, centered around the ideal distance d_{max}.

$$g(F_i, b_k) = \exp\left(-\frac{(d(F_i, b_k) - d_{max})^2}{2\sigma^2}\right) \tag{3.11}$$

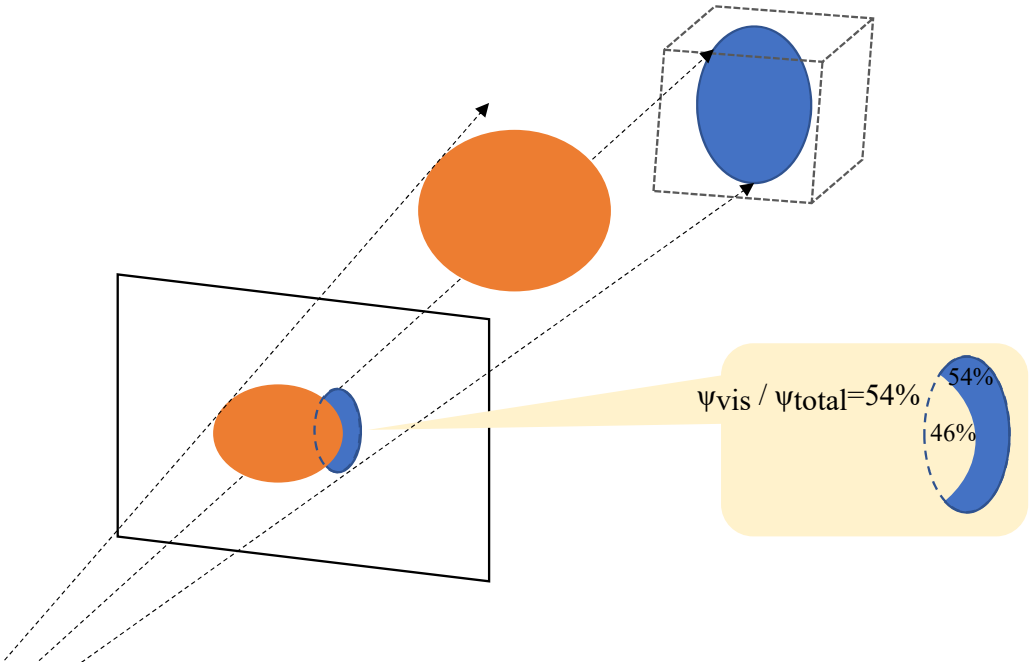


Figure 3.6: A view's contribution to a cell is weighted by the relative fraction of the cell's visible surface in the view.

The second term, the visible fraction of a cell (in pixel area units A_{px}), is determined as the ratio of the visible pixel count, ψ_{vis}, to the total pixel count. The visible pixel count is obtained by rendering a position buffer $\mathcal{P}(\mathbf{u})$ and obtaining a cell id b^{ID}. The cell id is generated by quantizing $\mathcal{P}$ with a factor q which is the chosen cell dimension and computing a spatial hash, such that $k = b_i^{ID}(\mathbf{u}) = h(\lfloor \mathcal{P}_i(\mathbf{u}) \cdot \frac{1}{q} \rfloor)$ if $\mathbf{u} \in b_k$. Using b^{ID}, we can easily determine a visible pixel count ψ_{vis} per cell, i.e., the visible pixels inside b_k:

$$\psi_{vis}(F_i, b_k) = \sum_{\mathbf{u} \in F_i} \delta(b_i^{ID}(\mathbf{u}), k) \tag{3.12}$$

The total pixel count is obtained by projecting the total surface area $A(b_k)$ of the scene geometry contained in cell b_k from world space into F_i (Figure 3.6). To avoid an exaggerated influence of very densely or very sparsely populated cells, we constrain the value to lie in the interval from one pixel, A_{px} to the projection of a cell face area A_{cell} into the view. Then, we convert from world-space area into pixel area units by normalizing with A_{px} to obtain a total pixel count ψ_{total}.

$$\psi_{total}(F_i, b_k) = \frac{1}{A_{px}} \cdot \max\left(\frac{\min(A(b_k), A_{cell})}{d^2(F_i, b_k)}, A_{px} \right) \tag{3.13}$$

Now we weight the pixel count by distance quality and cell coverage to obtain the positional coverage

$$\varphi_p(F_i, b_k) = g(F_i, b_k) \cdot \frac{\psi_{vis}(F_i, b_k)}{\psi_{total}(F_i, b_k)} \tag{3.14}$$

Candidate view evaluation. We use the coverage metric φ to decide if we add a candidate view F_i to F or not. To this end, we seek to increase the summed coverage τ_{sum} over all views and cells. Per cell, the coverage is clamped by a constant τ^{max} to avoid a bias towards cells covered by many views.

$$\tau(F, b_k) = \sum_{F_i \in F} \varphi(F_i, b_k) \tag{3.15}$$

$$\tau_{sum}(F) = \sum_{k} \min(\tau(F, b_k), \tau^{max}) \tag{3.16}$$

We keep a candidate image I, if it improves the coverage by at least $\Delta_{\tau_{sum}}$, i.e., $\tau_{sum}(F \cup I) > \tau_{sum}(F) + \Delta_{\tau_{sum}}$. When the view store is full, I must replace a victim v. By replacing every existing view F_i with I and finding the optimal coverage, we determine the victim $v = \arg\max_{F_i \in F} \tau_{sum}(F \cup I \setminus F_i)$. For efficiency, we keep the views in a list sorted by coverage, and only consider victims that are among the lowest-ranking views. If no victim v can be found such that τ_{sum} is increased by at least $\Delta_{\tau_{sum}}$, I is not kept.

3.1.3 View Processing

Before we store a candidate frame I, we must match its exposure to the existing views and ensure that its viewpoint is as accurate as possible, so that ghosting artifacts resulting from reprojection errors are minimal (compare images in Figure 3.7).

Using a subset of surface points $\mathbf{p}_j$ with $v(\mathbf{p}_j, I) = 1$ as sample points [42], we determine an exposure correction factor E_I. The exposure correction scales a new view I such that it best agrees with the median value of the other views $F_i \in F$ (each scaled with an exposure correction factor E_i) according to a robust metric m (such as the Tukey estimator). We use the median for robustness against outliers that come from observing specular reflections. The energy function $J(I)$ describes the agreement among measurements:

$$J(I) = \sum_{j} m\Big(E_I \cdot C_I(\pi(\mathbf{p}_j, I)) - \underset{i}{\text{median}}(E_i \cdot C_i(\pi(\mathbf{p}_j, F_i))\Big) \tag{3.17}$$

By minimizing $J(I)$, we obtained the desired exposure factor $E_I = \arg\min_{E_I} J(I)$. After obtaining

Figure 3.7: Quality comparison of "city wall" scene, from top to bottom: ground truth, lumigraph rendering with view processing (pose and exposure refinement), lumigraph rendering without view processing. In the latter case, blurriness and exposure differences reduce image quality.

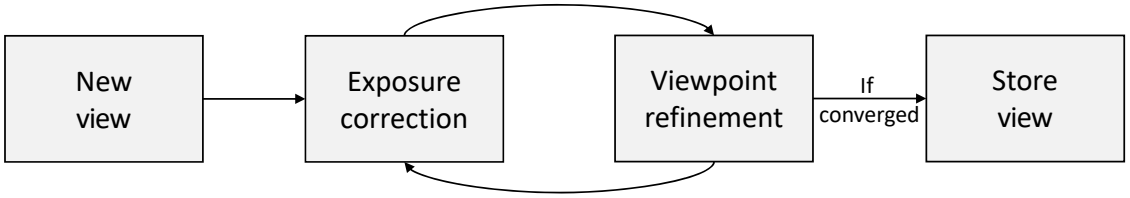


Figure 3.8: Before a new view is accepted into the view store, exposure and viewpoint are refined in an alternating optimization.

an initial estimate for the exposure correction of I, we rectify small errors in the viewpoint associated with I by making small changes to the external camera parameters, $p(\mathbf{M}_I)$ and $\hat{d}(\mathbf{M}_I)$ and recomputing $J(I)$. A search for a local minimum of $J(I)$ using the method of Farnebäck [30] determines the optimal camera pose $\mathbf{M}_I = \arg\min_{\mathbf{M}_I} J(I)$. We optimize exposure compensation and pose correction until convergence [111] (Figure 3.8). If the geometric reconstruction used as input contains views with pose outliers, the optimization can get stuck in a local minimum that can be detected by thresholding the residual error; such outlier views are discarded.

3.1.4 View Selection

View selection is composed of two parts, a pre-selection part computed every time a new view is accepted into the view store, and a final selection part executed in the fragment shader during view blending.

Pre-selection fills the references $R(b_k)$ when a new view I arrives. We add I to $R(b_k)$ if the cumulative coverage per cell is at least increased by Δ_τ, i.e., we make sure that $\tau(R(b_k) \cup I) > \tau(R(b_k)) + \Delta_\tau$. If $R(b_k)$ is full, we determine a victim to be replaced with I in $R(b_k)$ as $v_k = \arg\max_{F_i \in R(b_k)} \tau(R(b_k) \cup I \setminus F_i)$. If $R(b_k)$ is not yet full, $\Delta_\tau = 0$.

During view blending, the fragment shader iterates over the $R(b_k)$ and uses the validity function to determine if a particular view should contribute to the lumigraph at the given location or not. Out of the remaining views, the $n = 3$ best ones are used to determine the color of a pixel as described in section 3.1.1.

3.2 Evaluation

We integrated a prototype view planner into the Unity3D game engine on a desktop computer (CPU: Intel i7-5820K 3.30GHZ, GPU: Nvidia GTX 1080Ti). For best performance, the entire pipeline is

executed directly on the GPU, using HLSL shaders, while interface programming was done in C# for easy testing. The implementation expects a triangle mesh and an image sequence, annotated with camera poses, that can come from any (real-time or non-real-time) reconstruction algorithm. This allows us to conveniently test our system with a variety of reconstruction engines. We acquired the following four test scenes:

1. Viking village: synthetic scene with mostly diffuse materials
2. Apartment: synthetic scene with specular materials
3. City wall: real scene reconstructed using the multi-view stereo algorithm of Fuhrmann et al. [35]. Reconstruction software and the Darmstadt city wall dataset created by Fuhrmann et al. [35] is used to generate camera poses and proxy geometry.
4. Lab: real scene scanned with an RGB-D camera and geometrically reconstructed using InfiniTAM [137] [65]. Camera poses and proxy geometry are generated using implementation from Weilharter et al. [137].

For the synthetic scenes, we generated 5000 input images by rendering at poses densely sampled from a camera trajectory $H(i)$, $i \in [0,1]$, created by a 2D Hilbert curve as shown in Figure 3.9. For the real scenes, we used the original image sequences as input to our view planning algorithm.

We used this setup to analyse how various system parameters influence the results. We evaluate visual quality by comparing rendered views to reference images, reporting image quality as mean SSIM [132, 136], or MSSIM, for 200 test views taken with random camera poses. Reference images are generated using ground truth poses and geometry using synthetic datasets. Finally, we compare image quality of our method to a state-of-the-art real-time method and a commercial offline method.

3.2.1 Coverage Computation

Our first evaluation of the view planning method focuses on the coverage computation. Ideally, view references of a cell should already be full, before we run out of space in the view store.

To understand how much geometry is often (or never) seen in any view, we implemented an in-engine visualization tool which color-codes various aspects of the cells or the contained geometry (Figure 3.9), such as the number of views observing the cell, the number of registered views, the average coverage per registered views and the directional coverage. For example, it distinguishes geometry that is not seen by any input view from geometry that is seen in an input view, which is not selected for the view store. Additionally, to assess spatial behavior of our per-cell view planning,

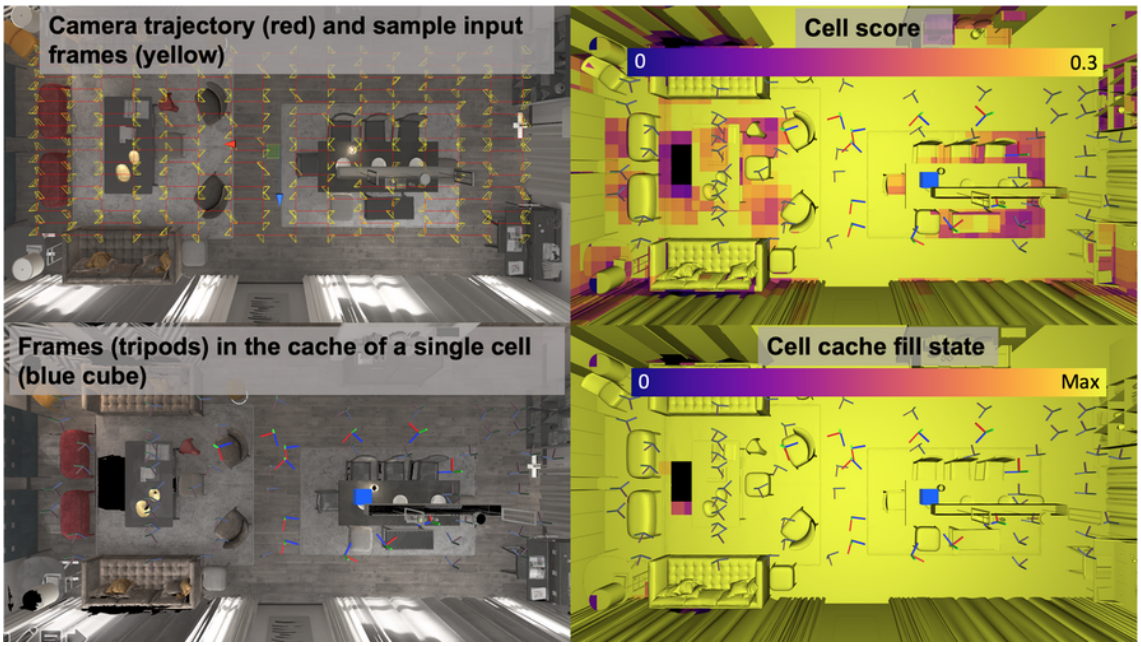


Figure 3.9: By using a trajectory following a Hilbert curve for generating the input image sequence in the apartment scene, we achieve a progressive, but homogeneous coverage (Top left). The coverage per cell is visualized (Top right). The homogeneous color indicates that all cells receive a relatively equal coverage in the view store. Note that to increase visual contrast at low scoring cells, scores have been clamped at 0.3. Coordinate glyphs mark the camera poses of the views referenced by the blue cell in the middle (Bottom left). The number of references stored per cell is visualized (Bottom right). Most cells fill all their references (yellow), while only a few inaccessible areas do not get covered properly (dark purple).

we visualize the selected and rejected views as small coordinate axis icons around a particular cell (Figure 3.9 and 3.10). These visualizations are generated in real-time and could be used during actual scanning, for example, with a handheld RGB-D sensor.

3.2.2 Trade-off Between Positional and Directional Coverage

We studied how blending of directional and positional coverage terms affects our rendering quality over two different types of trajectories. Using viking village, we compared a Hilbert curve fly-over trajectory (simulating a drone) to a walking-like trajectory with images taken at human eye-level. Each trajectory consisted of 5000 images overall.

Unlike the fly-over, occlusion varies significantly during the walking trajectory. One view may look down an entire street, while another one is complete occluded by a building. The walking trajectory benefits more from an increased weight given to positional coverage (Figure 3.11).

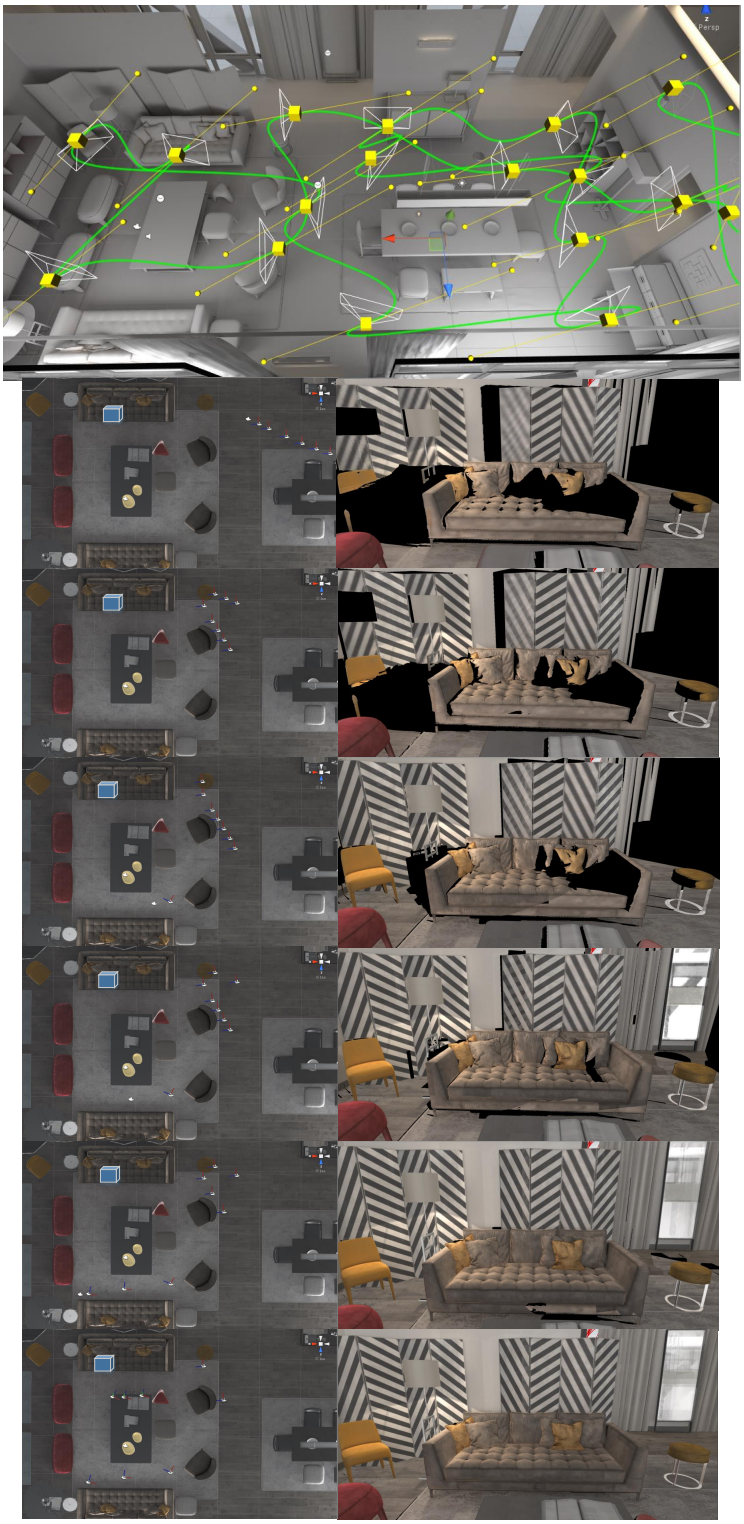


Figure 3.10: Image at the top visualizes an arbitrary input camera trajectory. Under the camera trajectory image, series of images side by side show progression of the key frame selection in the cache of a single cell (blue cube) and the image rendered using the cashed images.

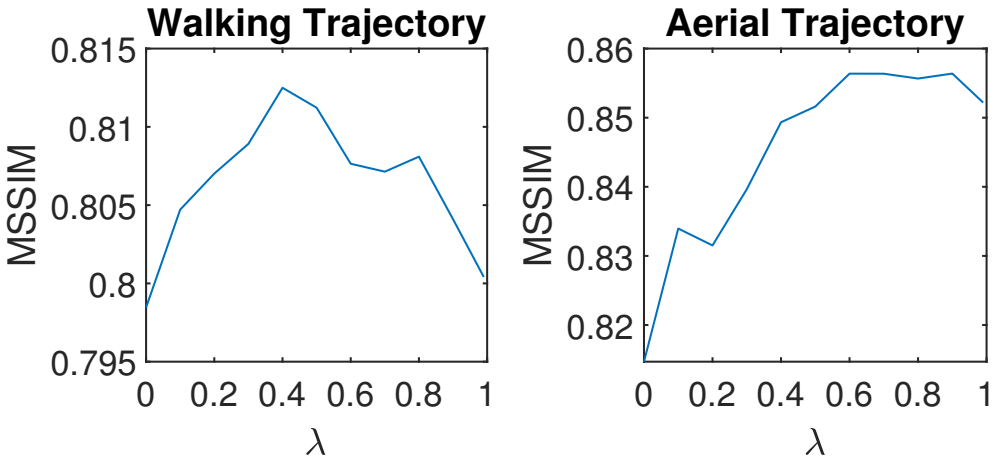


Figure 3.11: Effect of changing the weight of directional and positional coverage on MSSIM. The walking trajectory has more occlusions and requires careful choice of positional coverage (e.g., $\lambda = 0.4$).

3.2.3 Cell Dimension

We systematically varied the cell dimension and investigated how it influences the quality. Quality was measured as SSIM between images rendered using our system and reference views, averaged over 200 randomly chosen camera poses per test scene. Consequently, we report MSSIM over all views. We varied cell dimension depending on the overall scene size and plotted the results in Figure 3.12.

We observed an optimal cell dimension that depends on the size of the scene. Too small cell dimensions can cause a lower SSIM, since the views only have finite resolution, and varying views across very small cells encourages mosaicing artifacts. If the cell dimension gets too large (larger than 30-100 cm for the tested scenes), many views will not cover the entire area inside the cell and, in unfortunate situations, parts of the geometry are not properly covered. A heuristic based on the (expected) scene diameter is therefore a good solution.

3.2.4 Distance of Views and Cells

The parameters d_{max} and σ define the ideal distance of a view from a cell, such that the view covers the cell at the desired resolution. As can be seen in Figure 3.13, the choice of these parameters is not very sensitive in the apartment scene, which has a small overall diameter, so that even the furthest parts of the scene are covered with good resolution in every view. In contrast, the much larger viking village scene slightly benefits from an appropriate parameter choice. Here, a choice of $\sigma = 20m$ yields the best distribution of views with respect to the obtained image quality.

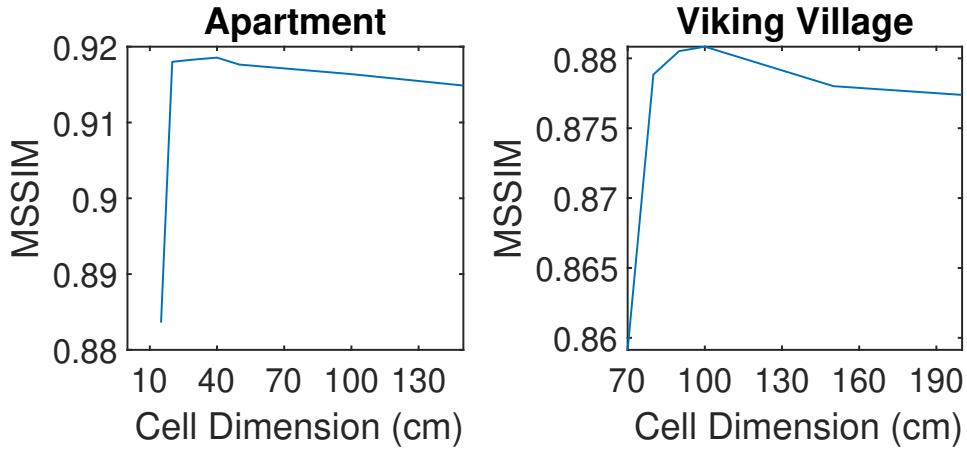


Figure 3.12: Effect of cell size on MSSIM. We observe the optimal cell distance depends on the scene diameter, in our examples, 30-100cm.

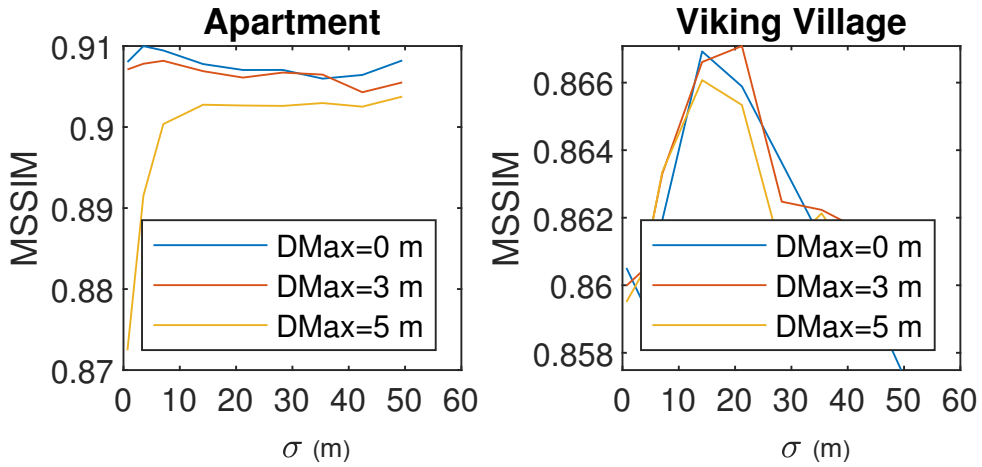


Figure 3.13: Effect of view-cell distance parameters σ and d_{max} on MSSIM.

3.2.5 View Store Size

We wanted to find out how the number of stored images (a few hundred 2 MPix images fit in GPU memory) influences quality. We processed the images using various cache sizes and plotted results for both datasets in Figure 3.14. Obviously, a larger cache contains more information and can yield a better image quality if used properly. Nonetheless, we observed that cache sizes above 100-200 images (again, dependent on scene characteristics) yield diminishing returns, implying that a finite number of views is sufficient, and a view store of reasonable size can be filled and maintained incrementally.

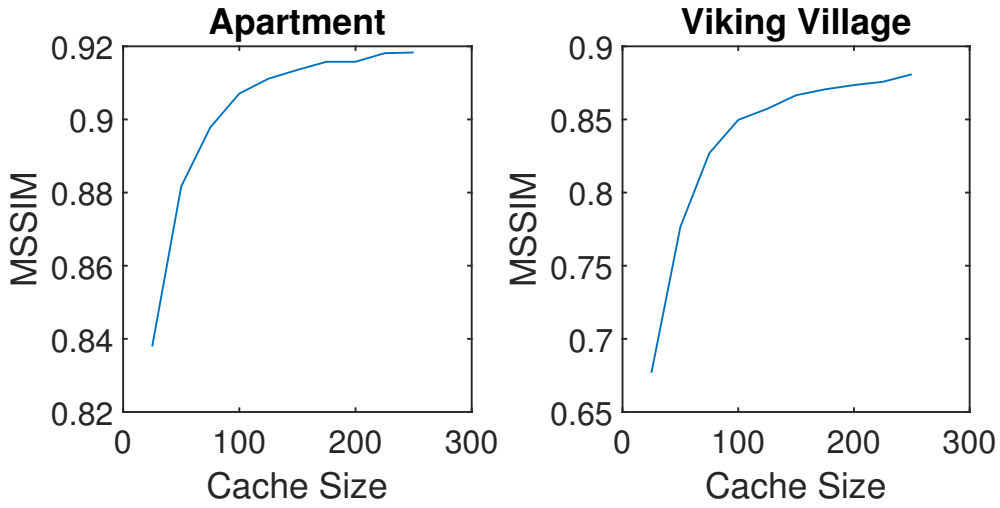


Figure 3.14: Diminishing returns in terms of image quality can be observed at reasonable caches sizes of 100-200 frames. MSSIM above 80% typically indicates that rendered images are subjectively highly similar to the ground truth images rendered from the same viewpoints.

3.2.6 Influence of Per-Cell View References on Quality

We studied how image quality changes with varying number of references per cell as shown in Figure 3.15. Note that all lumigraph renderings were produced by blending only three views per pixel, but chosen from the all references stored in a cell. More references per cell increase the chance that views with good coverage of a sample within that cell are found. However, an increasing number of references exhibits diminishing returns after 15-20 references, implying that cells can be properly covered with a small number of views, if chosen carefully. This insight is of practical importance, since the rendering speed depends on the reference number that must be searched in the fragment shader (Table 3.1). A reference number of 16 appears to be the best trade-off.

3.2.7 Influence of Refinement on Quality

We investigated the influence of view refinement on quality by computing the difference between lumigraph rendering results with refinement, without refinement, and the reference images. We present side by side images of before and after image pose refinement in Figure 3.7. As expected, image quality is significantly increased by the refinement. This result is consistent with observations made by other authors [89, 111]. However, our work demonstrates that a refinement carried out in real time (i.e., without extensive global search and optimization) is feasible.

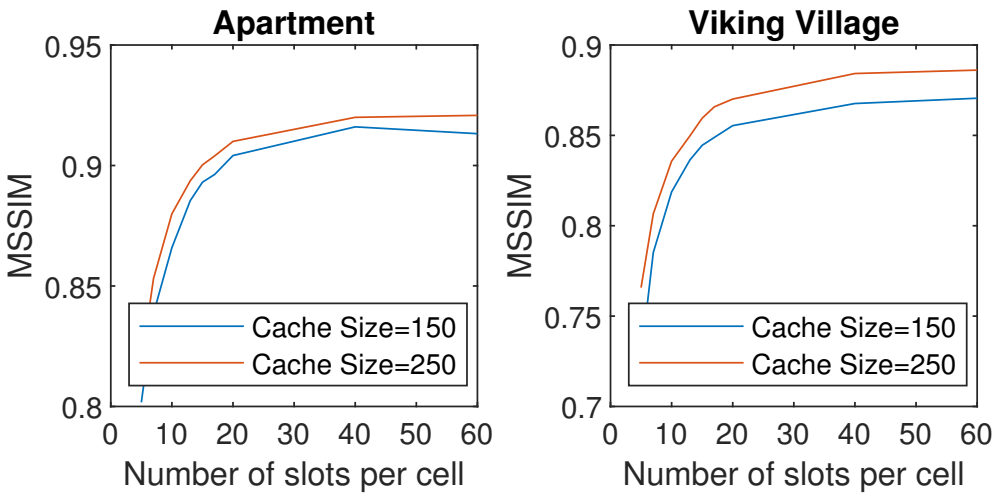


Figure 3.15: Effect of number of view references per cell on MSSIM. Diminishing returns can be observed after 15-20 view references.

Table 3.1: Performance overview. The left block of columns lists model name, primitive count (in million triangles) and input image resolution (in mega-pixels). The block with title "Model" gives timings of the modeling stage, and the block "Render", of the rendering stage, both in milliseconds. Column "Refine" is the average time for refining an input view. Columns labeled "ref" list timings for 8, 16, 32 or 60 references per cell, for either modeling or rendering.

Dataset			Model (ms)				
Name	Tris	Res.	Refine	8 ref	16 ref	32 ref	60 ref
Viking village	6.1M	2.1MP	-	4.2	4.5	6.3	22.7
Apartment	1.6M	2.1MP	-	3.1	3.1	8.7	16.9
City wall	10.3M	3.0MP	350	37.5	53.5	70.4	104.2
Lab	1.6M	0.3MP	60	2.1	15.2	44.2	63.3

Dataset	Render (ms)			
Name	8 ref	16 ref	32 ref	60 ref
Viking village	16.6	29.6	61	101.8
Apartment	16.5	19	43	89.3
City wall	23.6	37.3	69	109
Lab	16.6	17.7	29.8	45.9

3.2.8 Speed

We measured the performance of our algorithm by means of view selection speed and rendering speed. Specifically, we investigated performance impact of number of references as shown in Table 3.1. The performance reported in the table is obtained while rendering at 1080p resolution along a trajectory through the scene. Performance depends on both the geometric complexity of the scene

(which is out of scope of our work) and the number of references. At the recommended number of 16 references, we observe between 3-54 ms per input image for view planning (considering a new frame for the view store) and 18-37 ms for rendering. There is ample room for optimizations, but we already achieve interactive performance that would be suitable, for example, for viewing a remote scene in a telepresence application.

3.2.9 Quality Comparison to Reference Methods

We visually compare our method to two reference methods on the same input data: The first method is KinectFusion, which accumulates depth images in a volume tabulating a Truncated Signed Distance Function (TSDF) and extracts a mesh with per-vertex averaged colors. It represents the class of real-time methods. The second method used for comparison is 3DF Zephyr [1], a leading commercial photogrammetry software. It represents the class of offline methods, which can perform arbitrary global optimizations. Zephyr delivers noticeably higher precision geometry than KinectFusion, but required 40 min of compute time on the Lab dataset. Texture quality is optimized as well; however, Zephyr combines texture extracted from multiple images into one final surface texture, which is not view-dependent like ours. For a fair comparison, our method used the geometry delivered by KinectFusion under real-time constraints as input, and not the more precise geometry of Zephyr.

Side-by-side image comparisons can be seen in Figure 3.16. Informally, our method performs on par with 3DF Zephyr, while our method clearly outperforms the result delivered by KinectFusion in the same instant timeframe. KinectFusion's per-vertex coloring exhibits blurry rendering due to lack of high frequencies in its textures. While Zephyr utilizes textures much better, it has no intensity correction or view-dependent representation and shows significant mosaicing artifacts. Visual inspection of magnified details reveals that Zephyr's advantage of more precise geometry is lost if mosaicing cannot be suppressed.

To demonstrate the influence that the view planning method alone has on visual quality, we tested the image quality that our incremental view planning delivers compared to a globally optimal version of the same planning. We chose the 250 best out of 5000 keyframes by global optimization over all 5000 keyframes simultaneously, and compare the resulting image quality to the online method (which considers the views in strict sequence, one at a time, and not simultaneously). The globally optimal version of our algorithm results in an MSSIM increase of 1.3% compared to online version. Visual differences are only visible when looking carefully, as can be seen in Figure 3.17.

[1] http://www.3dflow.net

KinectFusion's rendering | Our rendering | 3DF Zephyr's rendering results

Figure 3.16: Comparation of our method to per-vertex colored KinectFusion model and to an offline-reconstruction rendering using a leading commercial photogrammetry software (3DF Zephyr).

3.3 Limitations

Despite the eye pleasing rendering of our work, we have identified a few limitations of our algorithm where it may not behave as expected. In this section we present these problems and possible solutions to it.

Mosaicing problem When completely different sets of keyframes are used by neighboring cells, this may cause the neighboring geometry to be rendered with slightly different intensity. Despite our algorithm adjusting intensity between keyframes, it does so globally not locally. As a future work, local patch based adjustment of the keyframe intensities could be implemented.

Pose correction Our algorithm corrects the poses of the keyframes in relation to the previous keyframes. Therefore, error can accumulate over time. This could be more problematic if the first keyframe's pose is already wrong. Future work can address this problem by readjusting the poses once the *SLAM* system detects a loop and updates the poses.

Figure 3.17: Rendering results with our incremental view planning algorithm (top) and the globally optimal version of our algorithm (bottom) are shown. Subtle differences can be realized only in a careful examination.

Keyframes with small field of view As an expected limitation, in datasets where no two keyframes observe the same part of a geometry, our algorithm won't be able to use the directional coverage criterion.

Scene dependent parameters We experimented with a limited number of datasets, but more complicated datasets such as indoor scenes with lots of object clutter, streets with narrow field of view or aerial drone footage with possible occlusions by mountains are possible. The parameters used in our algorithm have to be adjusted depending on the scene type. As a solution, the algorithm could recognize the type of scene using the visibility information in the keyframes and adjust the parameters dynamically instead of using predetermined ones.

3.4 Summary

We have described a new method for online view planning in reconstruction applications that builds an unstructured lumigraph. The views for the lumigraph are selected such that they cover the entire scene well without having to revisit older views, which are not longer available in the view store. Such an approach has previously only been explored in next-best view planning in robotics, but with the difference that our algorithm has no control over where the human operator will move the camera to. As far as we know, we are the first to explore this situation, although it is highly relevant for mobile Mixed Reality (MR) applications. In Chapter 4 we will introduce our inpaint fusion method to fill in the 3D holes that are present in the reconstructions we have. Also we will demonstrate how it can be utilized for removing unwanted graffiti and objects as well.

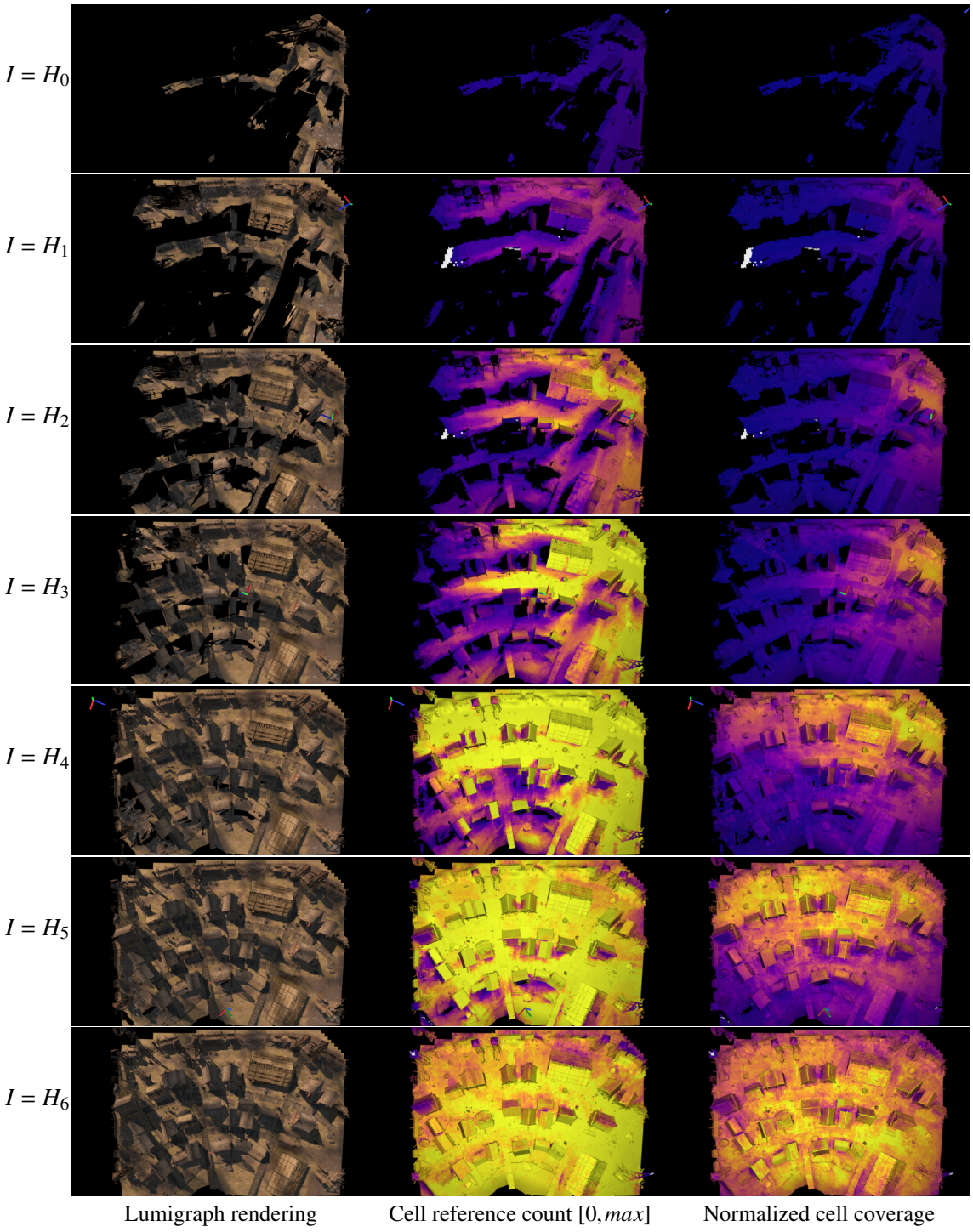


Figure 3.18: Visualization of view planning over time at *Viking village* dataset. H_x denotes the images rendered by sampling the Hilbert curve generated trajectory $H(i)$, $i \in [0,1]$ at regular intervals such that $\frac{\Delta i}{\Delta x} = \frac{1}{|H_x|-1}$ and $H_x \in [H(0), H(1)]$

4 InpaintFusion

Contents

In this chapter, we introduce our InpaintFusion algorithm, which is a state of the art algorithm in the field of Diminished Reality (DR). Traditionally, *DR* is implemented using a very coarse assumption over the object space. Assuming the inpainted region in the scene is sufficiently planar, previous work finds the dominant plane and renders it from different view points for inpainting. However, this approach is vulnerable to occlusions, which can easily exist even in a moderately complex scene. To overcome this limitation, our approach not only inpaints the RGB images, but also the depth. The depth map can be directly used to inpaint on top of Image-Based Modeling and Rendering (IBMR) generated images.

As covered in Section 1.5, using *DR* allows us to complement missing details of 3D reconstructions and hence facilitates the tele-exploration tasks. In addition to filling unobserved or unreconstructed parts of the scene, our algorithm also can remove unwanted details such as graffiti or 3D objects from *IBMR* results (Figure 4.1). At the end of this chapter, we show additional results to guide the reader through how InpaintFusion complements our *IBMR* system.

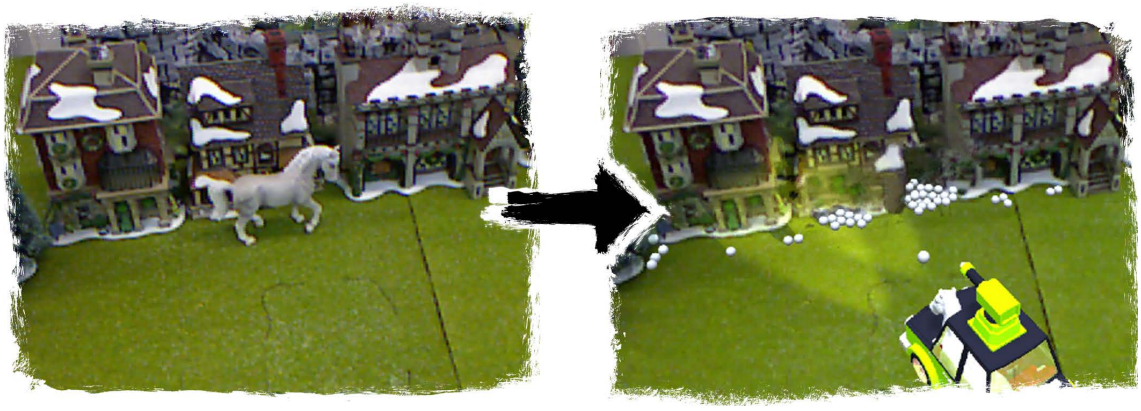


Figure 4.1: 3D diminished reality followed by augmented reality rendering. Our system, *InpaintFusion*, is able to remove objects and inpaint color and depth information. While previous attempts for real-time inpainting are limited to color channel inpainting only, *InpaintFusion* supports arbitrary depth channel inpainting as well. As it provides depth information, 3D augmented reality rendering in the inpainted environment becomes possible. This example shows a real white horse in front of a set of houses (left). We inpaint the color and depth values of the horse (right). This enables us to add a virtual car with head lights to illuminate the inpainted regions, together with white balls which can interact with the inpainted region by bouncing of the walls.

4.1 Method

In this section, we give an overview of our method, beginning with a concise problem statement.

4.1.1 The Problem of Depth Inpainting

Previous inpainting methods rely on homography warping and thus assume planar scenes [50, 123]. Once a keyframe F_0 is selected, the system inpaints the Region of Interest (ROI) as specified by the user. Given a homography relationship between F_0 and the current frame F_i, F_0 is transformed into the current frame as F'_0, and the system overlays F'_0 onto F_i. Since all the pixels in target region T_0 of F_0 are potentially visible in F'_0, the system can refine F'_0 using new pixel samples in F'_i with strong constraints on the patch appearance, in order not to break the texture [50], or progress with pixel searching in F_0 [72]. In other words, assuming the pixels' spatial relationship will be preserved by a homography transformation, these approaches can improve the inpainting over time regardless of viewpoint changes (see Figure 4.2).

For 3D inpainting, both update rules do not work. The 3D structure of the scene induces occlusions between the projected pixels of F_0 to F_i, causing new missing pixels to appear in F_i. These

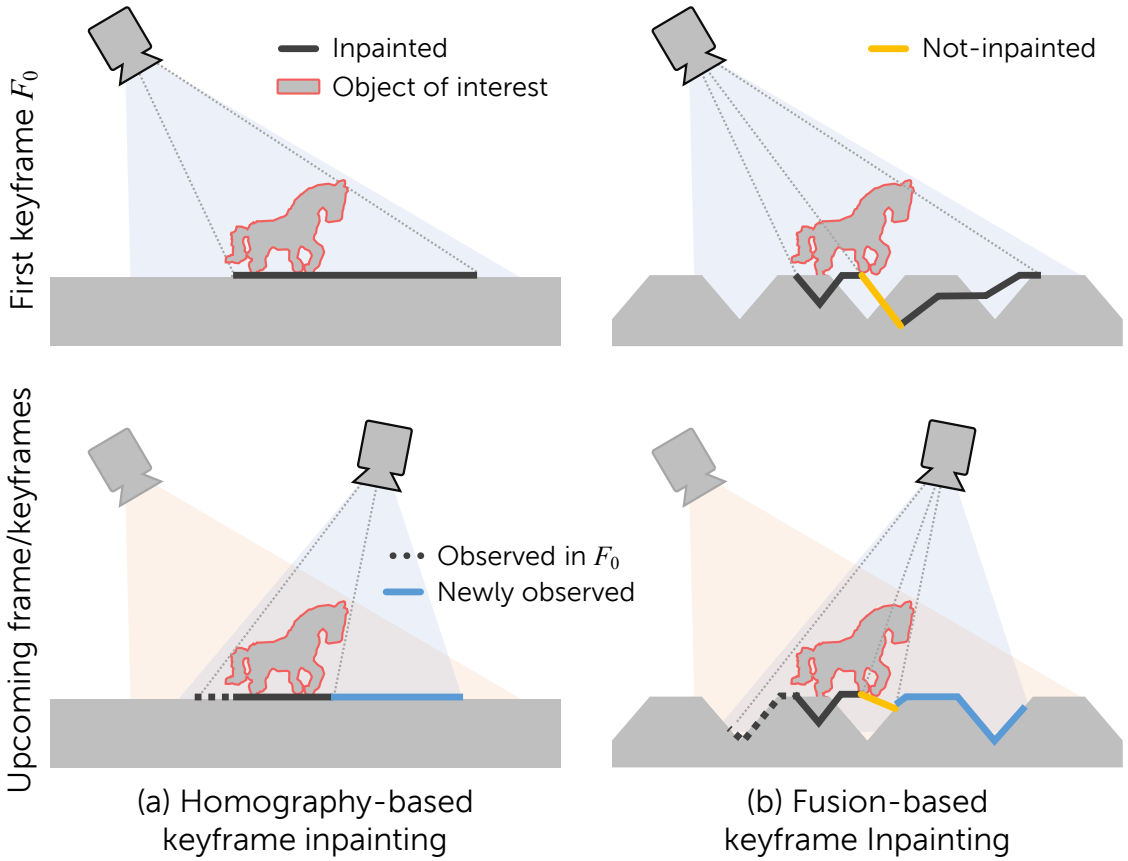


Figure 4.2: Comparison between homography-based keyframe inpainting and our multi-keyframe inpainting. (a) Assuming all pixels in a keyframe are visible, a homography to a single keyframe may be sufficient. This approach can even further update the keyframe over time by warping it. (b) Assuming a non-planar background, we fuse multiple keyframes. If the spatial relationship of the pixels changes due to occlusion, it is difficult to refine the pixels using the projected keyframe. We cannot project the current frame back to one of the keyframes, either, since currently visible pixels may not be visible in the keyframe anymore.

additional missing pixels needs to be inpainted in the projected keyframe F'_0.

Refining such hallucinated pixels does not work, since they have been collected from various sources, which are not necessarily consistent beyond the originally copied pixels. For this reason, previous work has resorted to manual labeling for additional constraints [50] or indirectly infering additional structure by decomposing the scene into multiple planes [72]. Refining F_0 does not resolve the problem, either, since the newly found missing pixels are mostly invisible at F_0 due to occlusion. Figure 4.2 depicts this problem.

Consequently, we need a novel approach for 3D inpainting, which incrementally fills in background

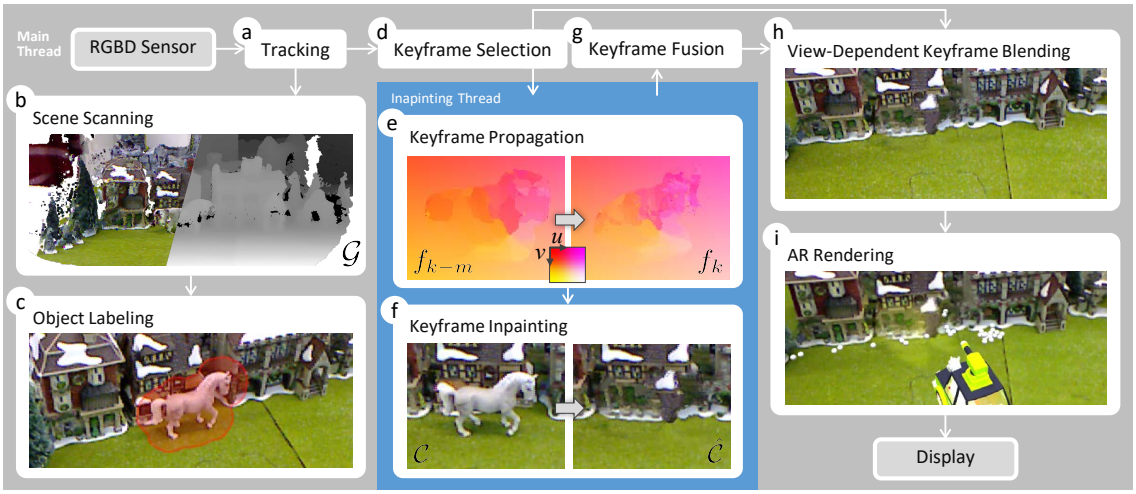


Figure 4.3: Overview of our *InpaintFusion* framework. Our method operates on an RGB-D frame. We first align the input frame in the world coordinate system by estimating the camera pose (a). This enables us to map pixels from the input frame to the 3D model and to a keyframe (b). Once the user labels the object of interest on a screen (c), the system preserves a keyframe (d) and inpaints the keyframe. We use previous inpainting results in keyframes to coherently map pixels over time (e). Subsequently, we search for an optimal set of pixel values which fill in the remaining unknown RGB-D values (f). To generate consistent 3D information, we fuse the inpainted depth map into a surfel map $\mathcal{G}$ (g). Finally, we blend available keyframes on the inpainted surfaces (h) and apply rendering effects to the 3D model (i).

3D structure without destroying previously inpainted color and structure. To this end, we propose to combine fusion from Simultaneous Localization and Mapping (SLAM) with multi-keyframe inpainting. A novel fusion method merges structural information of all inpainted keyframes into one consistent global map. We also present a rendering scheme to synthesize multiple inpainted color frames relying on labels from the *SLAM* system to minimize inpainted regions and to use observed background regions instead, if available.

4.1.2 System Overview

Our system supports per-pixel recovery of color and depth information in the unobservable region T in real time. For a frame F, it performs exemplar-based inpainting of a region T by copying information from $\overline{T}$, in both color and geometry domains, on top of a *SLAM* system [73]. Figure 4.3 illustrates our system architecture. Our system pipeline has the following stages:

Scene scanning using *SLAM*. We rely on a *SLAM* system to obtain an RGB-D frame $F_i = (C_i, \mathcal{D}_i, \mathcal{V}_i, \mathcal{N}_i, \mathbf{M}_i)$, consisting of a color buffer C_i, a depth buffer $\mathcal{D}_i$, a vertex buffer $\mathcal{V}_i$, a normal

buffer $\mathcal{N}_i$, and a 6DOF camera pose $\mathbf{M}_i$, expressed as an $\mathbb{SE}3$ transformation matrix. The frame F_i is fused over time into a global map $\mathcal{G}$ represented as a collection of surfels [107], i.e., the measurement F_i updates $\mathcal{G}$ in accordance with the previous work [73].

Object labeling. While *SLAM* runs, the user interactively labels a *ROI* in 2D screen space, which will be preserved in $\mathcal{G}$ and generate 2D target regions when projected.

Keyframe insertion. For stable inpainting over frames, we utilize keyframes. Our goal is to fill in all missing pixels in every novel viewpoint. Therefore, our system inserts a new keyframe when the pose diverges too much from the closest keyframe, while pixels remain missing in the *ROI*. Since the inpainting process is too costly to be completed before the next frame arrives, inpainting is performed in an asynchronous background thread.

Keyframe propagation. For visual consistency among inpainted keyframes, the transformation function f_{k-m} used for inpainting (see Section 2.2) the closest keyframe F_{k-m} is projected to a newly selected keyframe F_k to initialize the transformation function f_k. Here value of m is determined by the spatially closest keyframe's index. Instead of inpainting a surfel using only one keyframe, using pixels from other keyframes and blending them ensures a view dependent coloring. For the first keyframe F_0, as there are no other keyframes, the transformation function f_0 is initialized with random values (Figure 4.6).

Keyframe inpainting. An inpainted keyframe $\hat{F}_k = (\hat{\mathcal{C}}_k, \hat{\mathcal{D}}_k, \hat{\mathcal{V}}_k, \hat{\mathcal{N}}_k, \mathbf{M}_k)$ is computed for the new keyframe F_k by minimizing a cost function over all pixels $\mathbf{u} \in T_k$ with the given f_k, or, otherwise, from a random guess.

Inpainted keyframe fusion. The inpainted keyframe is passed to the *SLAM* system and fused with existing surfels. To avoid interference with tracking, inpainted keyframes are fused only in the *ROI*.

View-dependent keyframe blending. Labeled surfels are projected to the current frame F_i at $\mathbf{M}_i$ to obtain the *ROI* T_i, which is filled with pixels from multiple inpainted keyframes. The keyframes are projected to F_i via the fused inpainted surfels and blended depending on the viewpoint.

Augmented Reality (AR) rendering. Our system can provide the inpainted RGB-D frame or the inpainted global world model for additional *AR* rendering. In the following sections, we describe each of these stages in detail.

4.1.3 Scene Scanning Using *SLAM*

We use the dense map and device pose provided by the *SLAM* system of Keller et al. [73] to analyze the scene color and geometry, but extract some additional data from the global map $\mathcal{G}$. Each point in our global map $\mathcal{G}$ is represented as a surfel which contains a 24-bit RGB color $\mathbf{c} = [R,G,B]^{\mathrm{T}}$, a 3D position $\mathbf{p} = [X,Y,Z]^{\mathrm{T}}$, a normal $\mathbf{n} = [n_x,n_y,n_z]^{\mathrm{T}}$, a radius r, a depth confidence value $conf$, an index to distinguish a surfel among the rest of the surfels, and a label $l \in \{L_O, L_{IP}, L_{ROI}\}$, which classifies scene points as observed (L_O), inpainted (L_{IP}), or belonging to the *ROI* (L_{ROI}).

We smooth the sensor depth map $\mathcal{D}_i$ using a bilateral filter [128] and derive a vertex map $\mathcal{V}_i$ and a normal map $\mathcal{N}_i$. Subsequently, we estimate the pose $\mathbf{M}_i$ using an iterative closest point algorithm [86], aligning $\mathcal{D}_i$ with a virtual depth map, which we generate by reprojecting the global scene map $\mathcal{G}$ fused over time, into frame F_{i-1}. The inpainted region is not tracked; only surfels with L_O or L_{ROI} are used for generating the virtual maps. The vertex map $\mathcal{V}_i$ is derived from the smoothed depth map, and the normal map $\mathcal{N}_i$ is fused into the global map $\mathcal{G}$ using weights derived from the observed timing and the estimated confidence [73]. Initially, all the surfels are assigned the value L_O, until the user categorizes a surfel as L_{ROI}.

4.1.4 Object Labeling

One application of *DR* is removing undesirable objects, such as markers [71, 77, 122], pedestrians [84], cars [110], or any other category of objects that can be pre-trained [98]. Another type of application lets the user specify the *ROI*, e.g., by painting coarse strokes [50], and then segmenting and tracking these objects using image gradients. Gradient-based segmentation is fast and tends to work well on a planar background and a planar object of interest. In more complex environments, such as the multi-plane inpainting of Kawai et al. [72], interaction gets more complicated – the user must not only encircle the object, but also trim the segmentation when the view has changed too much, since the 3D object shape cannot be determined with sufficient accuracy from a sparse map from a visual *SLAM* system.

Since we have access to a dense map, our method can directly label the map by projecting the user's 2D input onto the depth map. Figure 4.4 shows an example of our labeling result. Inspired by incremental 3D segmentation [83], our system first encodes label information in pixels and then fuses the 2D label map into the global map. For this purpose, the user coarsely traces the object on a 2D screen to provide the input $\mathbf{u}_{user}$. Our system defines a 3D bounding disc with a center $\mathcal{V}(\mathbf{u}_{user})$, a radius r_{ROI}, and a thickness d_{ROI} along a surfel normal. Pixels that have vertices within

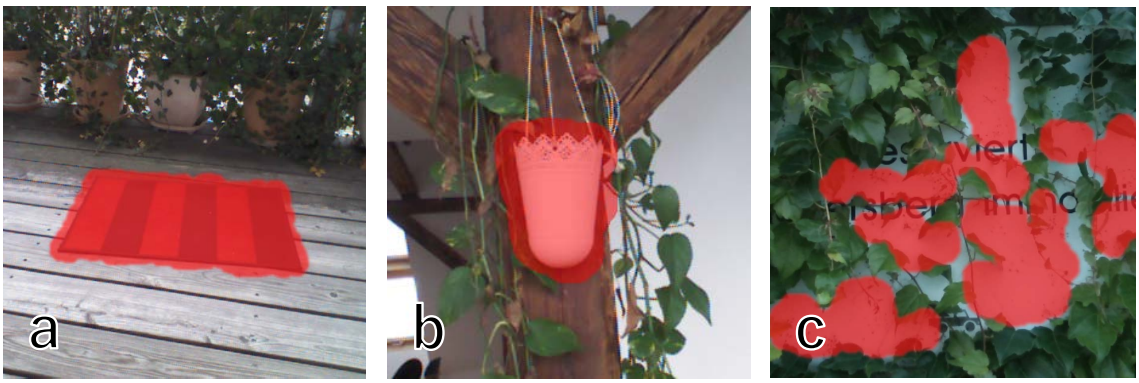


Figure 4.4: Labeling the 3D object of interest with 2D user interaction. The system translates user's pointing into a 2D label map, and fuses the label map into the global surfel map $\mathcal{G}$. Projecting such labeled surfel map to the tracked frame results in 2D registered label map for (a) a plane, or (b) a more complex object. (c) Our system allows users to label multiple objects.

the disc are labeled as L_{ROI}, and the rest, as L_O.

$$\mathcal{L}(\mathbf{u}) = \begin{cases} L_{ROI}, & \text{if } \|\mathbf{u}_{user} - \mathbf{u}\|_2 < r_{ROI} \ \wedge (\mathcal{V}(\mathbf{u}_{user}) - \mathcal{V}(\mathbf{u})) \cdot \mathcal{N}(\mathbf{u}) < d_{ROI} \\ L_O, & \text{otherwise} \end{cases} \tag{4.1}$$

The 2D label map $\mathcal{L}$ is fused with the 3D global map $\mathcal{G}$. To give safe margins for the *ROI*, the system dilates the region where such surfels are projected to the screen. The user can set a value for r_{ROI} that corresponds to the effective range of the labeling on the screen. For d_{ROI}, one may set a sensor depth uncertainty [73] that describes in which range the specified depth should cover surfels in the global map.

4.1.5 Keyframe Insertion

DR requires inpainting to be temporally coherent, which is usually addressed by using keyframes [50, 72, 123]. One can inpaint a frame as a keyframe and preserve it for future frames by warping the inpainted frame to the current frame. In the case of planar scenes, homography warping is sufficient as a geometric representation, as long as the *ROI* is tracked [72, 123] and pixel colors are referenced from visible regions within the frame [50]. In other words, pixel searching by inpainting need not to be repeated after inpainting the initial keyframe, and this strategy significantly reduces the processing time. In a planar scene, all inpainted keyframe pixels are potentially visible from any viewing angle.

In a scene with 3D structure, occluded pixels will occur within the *ROI*, as the camera moves

away from the keyframe; those pixels need to be inpainted (Figure 4.2). Our system inserts a new keyframe when the absolute pose difference from the closest keyframe F_k to the current frame F_i exceeds a threshold. The difference has been formulated as a Frobenius norm, i.e., $F_{\text{norm}}(i,k) = \|\mathbf{I} - \mathbf{M}_i^{-1}\mathbf{M}_k\|_{\text{F}}$. Given that our scale is in meters, we set the threshold to 0.7 which can be interpreted as in meters in case of neglegable rotation between $\mathbf{M}_i$ and $\mathbf{M}_k$ (Figure 4.1).

Additionally, our system checks the number of invalid pixels l', which do not have any labels, i.e., $l' \notin \{L_O, L_{IP}, L_{ROI}\}$. We denote regions of such pixels l' as $T_{l'}$. If $|T_{l'} \bigcap T|/|T| > \epsilon$ for a threshold ϵ (e.g., 0.1), the frame is selected as a new keyframe. In this way, we collect spatially distributed keyframes and safely exclude frames that do not observe target pixels to be inpainted.

4.1.6 Keyframe Propagation

When a keyframe F_k is inserted, we must ensure that inpainting at F_k is consistent among previous keyframes F_{k-m}. We address this requirement by initializing the transformation function f_k for F_k using the transformation function f_{k-m} associated with a keyframe F_{k-m}. We start with the closest keyframe, i.e., the keyframe with the minimum absolute pose difference and m is found by

$$\arg\min_m F_{\text{norm}}(k-m,k), \tag{4.2}$$

where F_{norm} is the Frobenius norm function to calculate pose difference between two keyframes. Figure 4.5 shows such an example keyframe propagation. Since keyframes remain stable over time, reusing the inpainted keyframes in the current frame generates temporally coherent results even for shaky camera motion, as long as the tracking works reliably. To bootstrap the mapping from F_{k-m} to F_k, we project $\mathcal{G}$ into F_k to provide an initial set of inpainted depth values for $\mathbf{M}_k$. Therefore, F_k contains mappings of L_{IP} and L_O in T and L_O in pixel sources S used for inpainting (see Section 2.2).

We transform F_{k-m} into F_k via geometric reprojection (i.e., forward warping), with the goal of reusing the pixel mapping stored in f_{k-m} on pixels in T_k. However, we avoid using the mapped pixel itself from the f_{k-m}. We are rather interested in the corresponding pixel in f_k as the inpainting in each image has to be consistent with the rest of the image. In addition, mixing pixel values from different images causes mosaicing problem. Therefore, we use $\mathcal{G}$ to calculate the transformation of image coordinates $\mathbf{u}_k \in \mathbb{R}^2$ of F_k into image coordinates $\mathbf{u}_{k-m} \in \mathbb{R}^2$ of F_{k-m}. These transformations are illustrated in Figure 4.6. We start by unprojecting the depth map $\mathcal{D}_k$ to 3D space,

$$\mathbf{p} = \mathbf{K}^{-1}[\mathbf{u}_k^{\text{T}}|1]^{\text{T}}\mathcal{D}_k(\mathbf{u}_k), \tag{4.3}$$

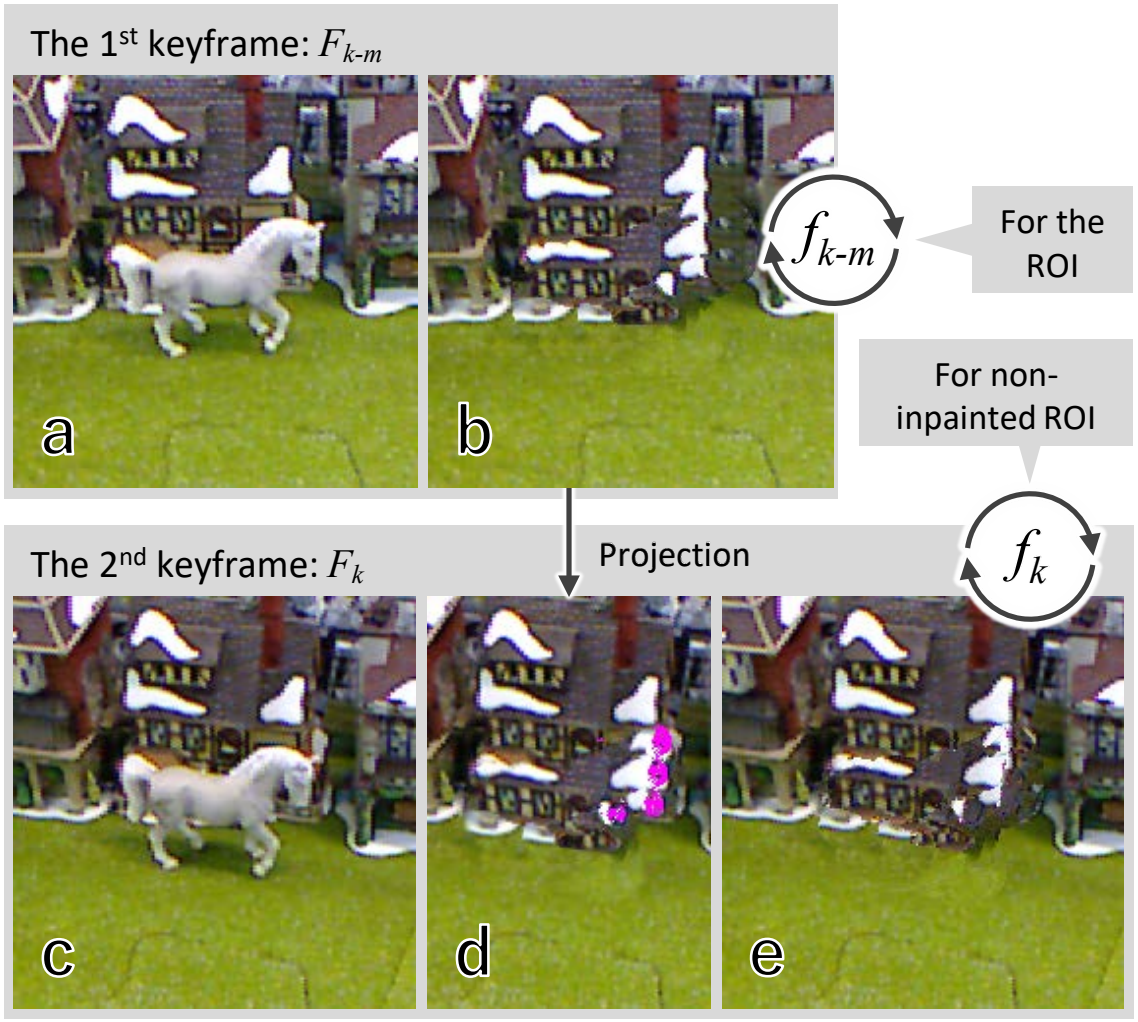


Figure 4.5: Spatio-temporally coherent keyframe inpainting. (a) The system selects a keyframe and (b) inpaints the keyframe. (c) Such an inpainted keyframe is projected to the newly inserted keyframe, where (d) non-inpainted regions may be revealed (magenta pixels). (e) Those pixels are newly inpainted to complete the inpainting in the new keyframe.

where $\mathbf{p} = [X, Y, Z]^{\mathrm{T}}$ is a point in the scene, and $\mathbf{K}$ is the 3×3 camera intrinsic matrix. After projecting a pixel into 3D space, the resulting point is further projected into the keyframe F_{k-m}:

$$\mathbf{u}_{k-m} = \pi([\mathbf{K}|\mathbf{0}]\mathbf{M}_{k-m}\mathbf{M}_k^{-1}[\mathbf{p}^{\mathrm{T}}|1]^{\mathrm{T}}), \tag{4.4}$$

where $\pi([X, Y, Z]^{\mathrm{T}}) = [X/Z, Y/Z]^{\mathrm{T}}$. Furthermore, we look up the inpainting results within the keyframe F_{k-m}. Thus, in F_{k-m}, we use the transformation f_{k-m} at the pixel located at $\mathbf{u}_{k-m}$. The

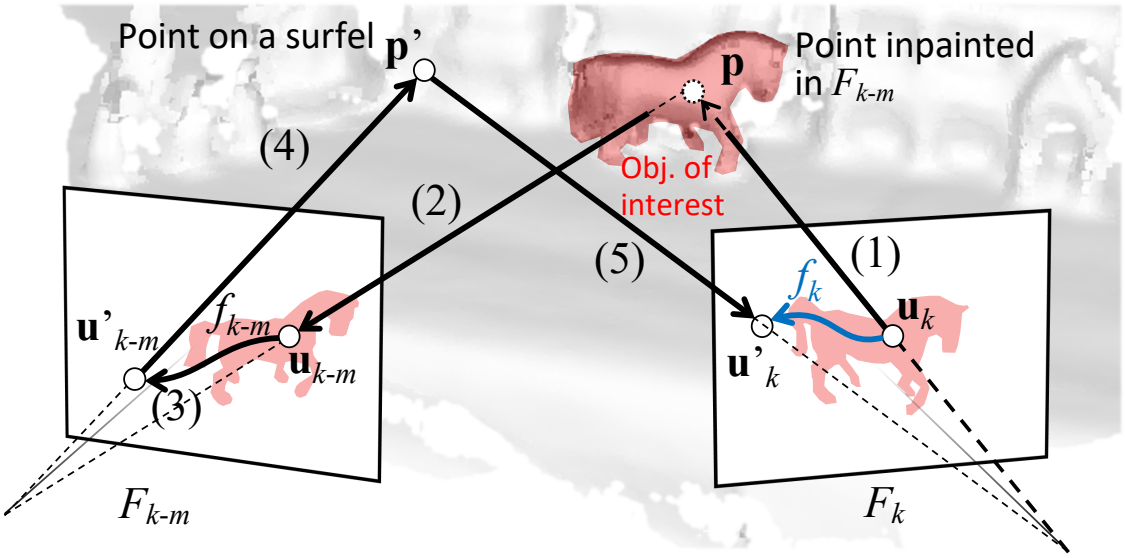


Figure 4.6: Keyframe propagation. As color of a surfel is view dependent due to light and matter interaction, we avoid inpainting surfels from one single keyframe. In addition, inpainting neighboring surfels using a mix of different keyframes, are also unacceptable as this creates a mosaicing problem. Therefore, for each new view point, namely the new keyframe, we inpaint surfels using the pixels of that keyframe. The mapping f_k (blue arrow) from $\mathbf{u}_k$ to the reference point $\mathbf{u}'_k$ is defined by a series of transformations (black arrows). (1) First, we project $\mathbf{u}_k$ to the world coordinate point $\mathbf{p}$ (2), before we project $\mathbf{p}$ into the other keyframe, which identifies $\mathbf{u}_{k-m}$. (3) At $\mathbf{u}_{k-m}$, we look up f_{k-m} to find $\mathbf{u}'_{k-m}$, and (4) we project $\mathbf{u}'_{k-m}$ back to the world coordinate systems onto $\mathbf{p}'$. (5) Projecting $\mathbf{p}'$ back into the current frame identifies $\mathbf{u}'_k$.

result is the reference position $\mathbf{u}'_{k-m}$ from which the pixel value was taken to inpaint the keyframe.

$$\mathbf{u}'_{k-m} = f_{k-m}(\mathbf{u}_{k-m}) \tag{4.5}$$

This provides a good guess for color and depth values based on the keyframe data F_{k-m}. However, to ensure intra-frame consistency, we are interested in selecting pixel values from the new keyframe F_k rather from the previously preserved keyframe F_{k-m}. Therefore, we cannot directly take the pixel values at position $\mathbf{u}'_{k-m}$. Instead, we are looking for the corresponding pixel position of $\mathbf{u}'_{k-m}$ in F_k. We compute the projection of $\mathbf{u}'_{k-m}$ into F_k, denoted as $\mathbf{u}'_k$. We derive this transformation by unprojecting the pixel to 3D space, followed by a projection of the corresponding 3D point into F_k.

The complete series of transformations required to map a 2D coordinate $\mathbf{u}_k$ to $\mathbf{u}'_k$ in F_k is given in Equation 4.6. Our approach is applied to all pixels within the *ROI* T_k.

$$f_k : \mathbf{u}_k \overset{Eq.\ 4.3}{\rightarrow} \mathbf{p} \overset{Eq.\ 4.4}{\rightarrow} \mathbf{u}_{k-m} \overset{Eq.\ 4.5}{\rightarrow} \mathbf{u}'_{k-m} \overset{Eq.\ 4.3}{\rightarrow} \mathbf{p}' \overset{Eq.\ 4.4}{\rightarrow} \mathbf{u}'_k \tag{4.6}$$

Figure 4.7: Comparison of similarity metrics. From left to right: original, linear combination of texture and geometry similarity, and multiplicative combination of texture and geometry similarity (our approach). The linear combination requires three parameters to be adjusted for inpainting, while the multiplicative approach requires one parameter.

If a projected point $\mathbf{u}'_k$ is exceeding the bounds of frame F_k, we assign a random 2D coordinate as the output of $f_k(\mathbf{u}_k)$. For multiple keyframes, we repeat the above mapping from the closest keyframe to the furthest one, until all the pixels in T_k are processed, or a pre-determined number of keyframes have been processed.

4.1.7 Keyframe Inpainting

Frame pre-processing. There might be missing pixels in a single depth map due to the limitations of the depth sensor, even though the depth map is projected from the global map. Since pixels without depth cannot be used as sources for inpainting, we require sufficiently dense depth to obtain plausible results. Thus, we first fill in missing pixel depth using convolutions [119] for edge-aware inpainting. After this densification, $\mathcal{V}_k$ and $\mathcal{N}_k$ are calculated again from the the depth map.

Finding reference pixels. The initial projection of $\mathcal{G}$ into F_k may leave some pixels in T_k uninitialized, so we need to fill in these unmapped pixels from scratch. We find the transformation f^* of the remaining pixels by using the PatchMatch algorithm [3] for minimizing the cost function in Equation 4.7. Similar to previous examplar-based inpainting, we model the overall cost ρ as a weighted sum of color (texture) similarity, ρ_t, spatial similarity, ρ_s, and a novel geometric similarity term, ρ_g:

$$f^* = \arg\min_f \sum_{\mathbf{u}\in T} w\rho_t(f,\mathbf{u})\rho_g(f,\mathbf{u}) + (1-w)\rho_s(f,\mathbf{u}) \tag{4.7}$$

The texture cost (Equation 4.8) minimizes the appearance difference between pixels to be inpainted in T and pixels referenced in $\overline{T}$, while the spatial cost function forces pixels in the area surrounding a target pixel to cluster, so that spatial continuity can be maintained (Equation 4.9).

$$\rho_t(f,\mathbf{u}) = \sum_{\mathbf{v}\in\{\pm1,\pm2\}^2} \|C(\mathbf{u}+\mathbf{v}) - C(f(\mathbf{u})+\mathbf{v})\| \tag{4.8}$$

$$\rho_s(f, \mathbf{u}) = \sum_{\mathbf{v} \in \{\pm 1\}^2} \|f(\mathbf{u}) + \mathbf{v} - f(\mathbf{u} + \mathbf{v})\| \tag{4.9}$$

In addition to texture and spatial similarity, we model a cost function ρ_g for the geometric appearance. In Equation 4.7, the geometric appearance term modulates the texture similarity ρ_t, acting like a weight that forces both texture and normals to agree. Adding ρ_g as another linear term to sum of ρ_t and ρ_s works as well [50], but it introduces one more additional weighting parameters that must be adjusted as three linear terms are combined. The resulting subtle differences are illustrated in Figure 4.7.

Since directly using depth buffer $\mathcal{D}$ would suffer from the perspective and view-dependent nature of a depth map, we use the normal map for inpainting instead. We derive a normal map from the depth values by using a gradient estimator. However, since the raw depth map suffers from noise and incomplete areas, the derived normal map will be affected as well. Therefore, we use $\mathcal{N}_G$ derived from the projection of the world space depth to the keyframe frame F_k, i.e., $\mathcal{N}_G \rightarrow \mathcal{N}_k$.

$$\rho_g(f, \mathbf{u}) = \sum_{\mathbf{v} \in \mathcal{N}_G} 1/\max(\kappa, \mathcal{N}_G(\mathbf{u} + \mathbf{v}) \cdot \mathcal{N}_G(f(\mathbf{u}) + \mathbf{v})) \tag{4.10}$$

Here, κ is a lower bound to avoid division by zero. Pixels having similar normals are clustered naturally. In other words, ρ_g provides a geometrical labeling that limits pixel search to geometrically similar surfaces, overcoming the need for manual labeling used in previous work [50].

The transformation map is randomly initialized when the first keyframe is registered to the system. From the second keyframe on, the closest keyframes' results are propagated as outlined in Section 4.1.6. We also take a coarse-to-fine approach to find f^* in reasonable time.

Generating pixel values. After finding references for all pixels in the *ROI*, we are able to copy the corresponding pixel values. To inpaint the color $\hat{C}$, we simply copy the values according to f^*.

$$\hat{C}(\mathbf{u}) \leftarrow C(f^*(\mathbf{u})) \tag{4.11}$$

Since we cannot simply copy view-dependent depth values, we use the normal map $\hat{\mathcal{N}}$ for inpainting 3D structure. The normals at reference pixels can simply be copied like color values.

$$\hat{\mathcal{N}}(\mathbf{u}) \leftarrow \mathcal{N}_G(f^*(\mathbf{u})) \tag{4.12}$$

For the depth values in the *ROI*, we compute $\nabla\hat{\mathcal{D}}$, a *gradient field* [106] (depth gradient map) of the sampled depth values. We minimize

$$\min_{\mathbf{u}\in T}\sum(\nabla\hat{\mathcal{D}}^*(\mathbf{u})-\nabla\hat{\mathcal{D}}(\mathbf{u}))^2 \tag{4.13}$$

to calculate the inpainted depth map $\hat{\mathcal{D}}^*$. Note that directly sampling pixels from f^* will introduce inconsistencies: Consider a pixel at $\mathbf{u}$ and its right neighbor at $\mathbf{u}+\mathbf{v}$. A naive horizontal gradient

$$\nabla\hat{\mathcal{D}}(\mathbf{u}) = d(f^*(\mathbf{u}))-d(f^*(\mathbf{u})+\mathbf{v}). \tag{4.14}$$

will usually not match the sampled depth gradient of the adjacent pixel, $d(f^*(\mathbf{u}+\mathbf{v}))-d(f^*(\mathbf{u}+\mathbf{v})-\mathbf{v})$. Therefore, we minimize

$$\min_{\mathbf{u}\in T}\sum(\nabla\hat{\mathcal{D}}^*(\mathbf{u})-\nabla\hat{\mathcal{E}}(\mathbf{u}))^2, \tag{4.15}$$

where $\hat{\mathcal{E}}(\mathbf{u})$ is the mean bi-directional depth gradient sample:

$$\nabla\hat{\mathcal{E}}(\mathbf{u}) =(d(f^*(\mathbf{u}))-d(f^*(\mathbf{u})+\mathbf{v})+d(f^*(\mathbf{u}+\mathbf{v})-\mathbf{v})-d(f^*(\mathbf{u}+\mathbf{v})))/2 \tag{4.16}$$

After recalculating the vertex and normal maps, $\hat{\mathcal{V}}$ and $\hat{\mathcal{N}}$, from the inpainted depth $\hat{\mathcal{D}}^*$, we obtain an inpainted keyframe $\hat{F}_k = (\hat{\mathcal{C}}_k, \hat{\mathcal{D}}^*_k, \hat{\mathcal{V}}_k, \hat{\mathcal{N}}_k, \mathbf{M}_k)$. Figure 4.8 shows an example of inpainting.

4.1.8 Keyframe Fusion

An inpainted keyframe $\hat{F}_k$ is fused into the global map to obtain a uniform, consistent representation $\mathcal{G}$. While the user is presented with inpainted frames, the *SLAM* tracking should not see the inpainting results containing hallucinated data. Therefore, we fuse the inpainted keyframe $\hat{F}_k$ only with surfels bearing the L_{IP} label, or we insert new surfels labeled L_{IP}, if the unprojected space is vacant. Such newly generated surfels are given a high conficdence value, to ensure that they appear immediately in the next frame. The *SLAM* tracking only sees surfels with L_O and L_{ROI}.

4.1.9 View-dependent Keyframe Blending

Basic blending function. We fill-in the *ROI* at the current frame F_i only from completely inpainted keyframes $\hat{F}_k$. As the surfel resolution in the current view may deviate significantly from the keyframe pixel resolution, calculating a blending weight per surfel can be computationally inefficient, since multiple pixels of $\hat{F}_k$ can be projected onto a single surfel, or one projected pixel can spread onto multiple surfels.

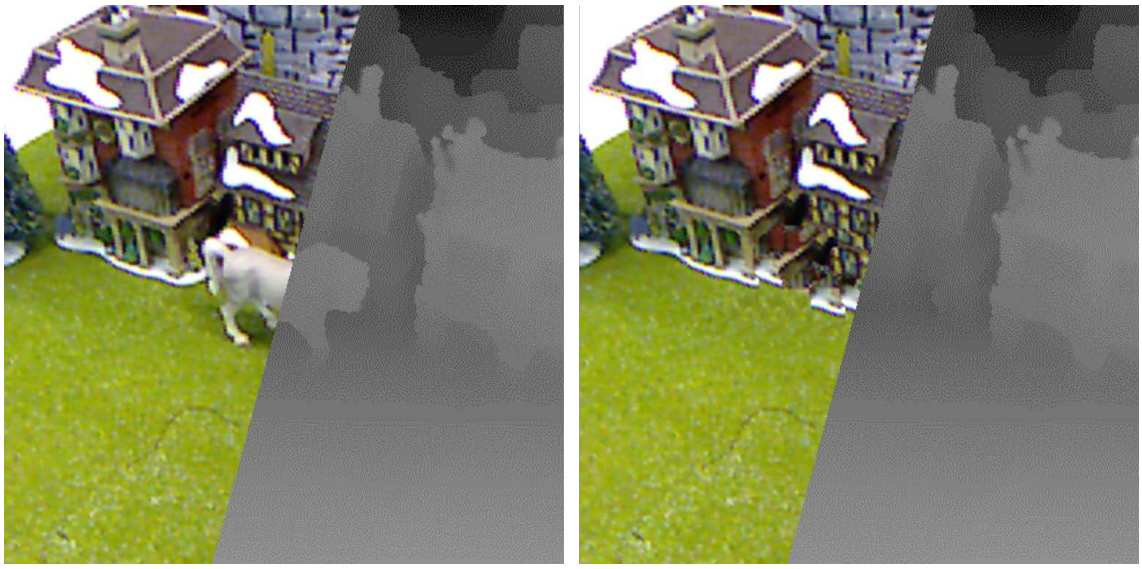


Figure 4.8: A complete 3D inpainting for a keyframe. (Left) Given an RGB-D frame where depth image is calculated using global map $\mathcal{G}$, the proposed method finds globally optimized pixels in the color and normal maps calculated from the depth map. (Right) The proposed method also calculates a depth map using sampled depth according to the optimized transformation map. The observable dithering effect on the calculated depth map is due to the different orientations of the neighboring surfels.

To avoid the expensive weight calculation at each point of the dense geometry proxy [10, 22], we calculate weights for the M closest keyframes instead and blend the M keyframes with the following weights for projected surfels of label L_{IP} and L_O:

$$w_k = \frac{\exp(-(d_k^{\text{APD}})^2)}{\sum\limits_{m \in \sum_{m=1}^{M}} \exp(-(d_m^{\text{APD}})^2)} \tag{4.17}$$

Combining observed and inpainted pixels. We always prefer observations over inpainted pixels by giving higher blending weights to such pixels. We distinguish between pixels that have been *observed* before, pixels that have been *inpainted*, and pixels in the *ROI*. As described in Section 4.1.3, we assign the corresponding labels, L_O, L_{IP}, and L_{ROI}. The label information is already available in each keyframe when it is projected before inpainting. For each pixel $\mathbf{u}$, we check if it is projected to the *ROI* in a new keyframe F_k:

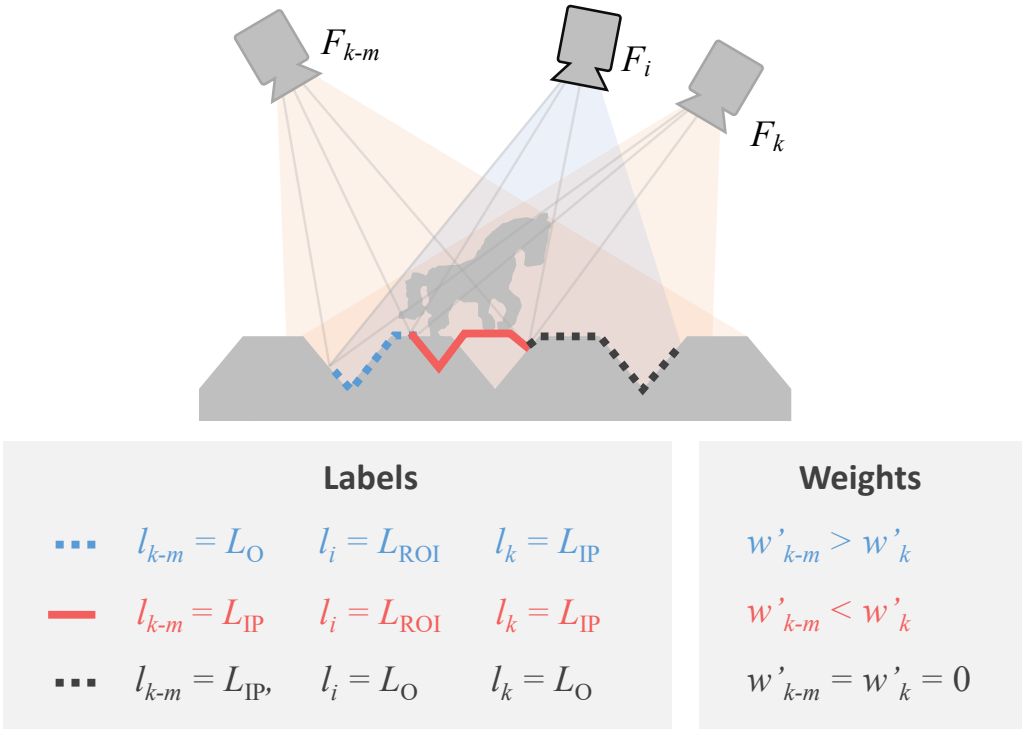


Figure 4.9: Distinguishing observed surface and inpainted surface by blending weights in the keyframe blending.

$$w'_k(\mathbf{u}) = \frac{\exp(-(d_k^{\text{APD}} w_o(\mathbf{u},0))^2)}{\sum_{m \in M} \exp(-(d_m^{\text{APD}} w_o(\mathbf{u},m))^2)} \tag{4.18}$$
$$w_o(\mathbf{u},m) = \delta(l_{k-m}(\pi(\mathbf{M}_{k-m}\mathbf{M}_i^{-1}\mathcal{V}_i(\mathbf{u}))), L_{IP}),$$

where δ is the Kronecker delta function. Figure 4.9 illustrates our categorization of pixel-wise rendering based on global map labels. First, the red and black regions of F_i are inpainted in F_{k-m}, and surfels are labeled as L_{IP}. The blue region remains labeled as observed L_O. Then, the *ROI* of F_k is inpainted, i.e., the red and blue regions, using the rendering of F_{k-m} in the blue region. However, the label of the blue region is set to L_{IP}, since it is synthetic. Therefore, only the red region is inpainted, receiving information from F_{k-m} as the initial guess for the inpainting. When rendering F_i, pixels in the blue and red regions must be inpainted. In our example, we have two keyframes and need to calculate their blending weights.

In the blue region, $w'_{k-m} > w'_k$, even though F_i is closer to F_k, since F_{k-m} is based on a sensor observation. In the red region, we only have surfels with L_{IP} labels. Therefore, the blending

weights depend on how close F_i is to the keyframes, i.e., $w'_{k-m} < w'_k$ in this case. The black region is a direct observation of F_i and is out of the range of keyframe blending. This procedure combines inpainting-based *DR* with observation-based *DR*, while minimizing the inpainting area.

4.1.10 AR Rendering

Since a full resolution color and geometry map $\hat{F}_i = (\hat{\mathcal{C}}_i, \hat{\mathcal{D}}_i, \hat{\mathcal{V}}_i, \hat{\mathcal{N}}_i, \mathbf{M}_i)$ from a global map is accessible at each frame, we can use various *AR* rendering methods for the frame after the inpainting. All the geometry-related maps correspond to G-buffers; therefore, *InpaintFusion* lends itself to any kind of deferred rendering. Figure 4.1 shows an example of relighting in the inpainted *AR* space.

4.2 Evaluation

We evaluate *InpaintFusion* concerning performance and quality in 3D scenes. Therefore, we implement three different types of geometry proxies on top of *InpaintFusion* to demonstrate how typical geometry proxies used in previous work affect the quality of the resulting 3D inpainting, while *InpaintFusion* can maintain its quality in various scenes without explicitly defining geometry proxies.

4.2.1 Quality Assessment

As inpainting has no ground truth, previous work in Table 2.1 does not present quantitative measures and typically relies on subjective preference. In fact, quantitative assessment in inpainting is an open research problem, as discussed by Isogowa et al. [62]. They propose an automated evaluation for inpainting methods that significantly reduces manual labor in the training step. Such automated quality assessment, however, will not be able to truly reflect human judgement. In addition, spatio-temporal consistency cannot be judged from individual images. For these reasons, we perform a study with human subjects.

The goal of our assessment is to demonstrate that *InpaintFusion* surpasses existing work in terms of subjective quality. To this end, we compared *InpaintFusion* to two other popular approaches, which are based on a geometry proxy. In particular, we compared results of *InpaintFusion* to those generated using a single plane [49, 50, 71, 77, 122] and a multi-plane approach [72, 123]. In the single and multi-plane approach, only one keyframe is inpainted, such that the resulting image looks plausible. Therefore, the quality in subsequent frames only depends on the warping of the inpainted keyframe to the current frame.

Figure 4.10: Comparison of different types of geometry proxies on the *Crack* dataset. (Top left) original, (Top right) single plane, (bottom left) multi-plane, and (bottom right) *InpaintFusion*. The insets in the lower left corner show depth encoded as greyscale values.

To generate the results with the single plane approach, we manually selected three points in the keyframe to define a plane. The geometry for the multi-plane approach is generated using the mean-shift plane estimator proposed by Kawai et al. [72]. For the estimated planes, we generated the corresponding depth map and inpainted the color channels. We estimated the camera pose over time with an RGB-D tracker that minimizes point to plane distances [101] in addition to color [124] residuals in image space, as implemented in OpenCV.

Hereafter, we refer to the contender methods as *Single Plane* and *Multi-Plane*, respectively. Note that these planes inpaint the depth before the color channels are inpainted. Also, such explicit geometry gives enough constraints in Equation 4.10 to inpaint each plane from geometrically separate pixels without the need for separate inpainting in each plane [72] or manual labeling [50]. For fair comparison in the quality assessment, we used the same tracker and mask images in all three methods including *InpaintFusion*.

4.2.2 Inpainting Quality

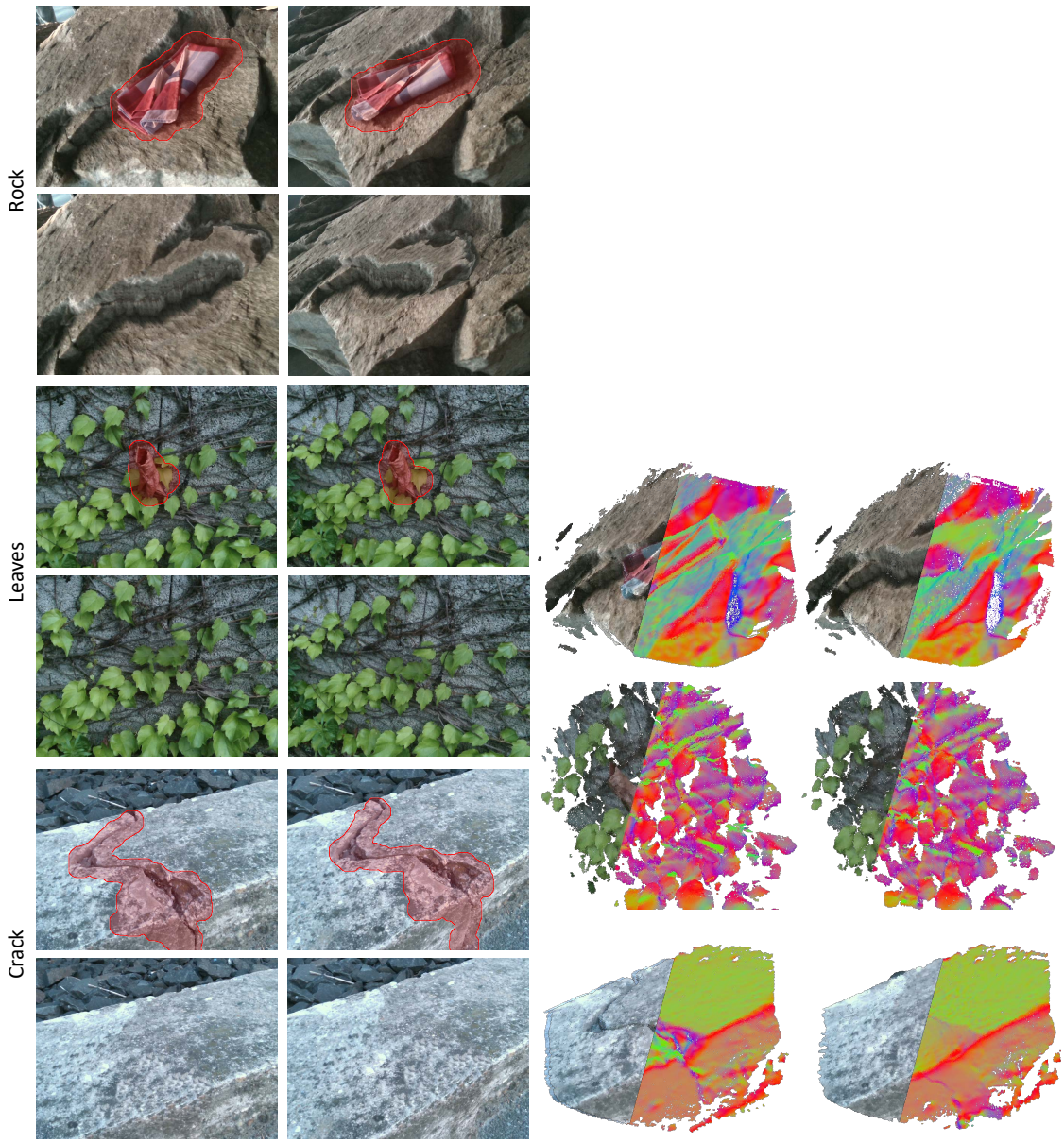
Figure 4.11 shows inpainting results in selected frames and fusion results of *InpaintFusion* in three scenes that have reasonable 3D structures: *Rock*, *Leaves*, and *Crack*. In *Rock*, we inpainted the handkerchief on a rock. The system generated a seamless but fake surface in 3D. Owing to the image-space color and depth inpainting and subsequent fusion of those images, the inpainted structure fits the 3D structure of the real rock even under significant viewport changes. In *Leaves*, we inpainted the dead leaf among the green leaves to replace the dead leaf with green leaves hallucinated by the proposed system. Note that real leaves are occluded by the generated leaves, and partially observable real leaves from different viewpoints retain their shape and color in each view. In *Crack*, we inpainted a crack on a rock to virtually fix the crack. Note how the inpainting kept the geometric edge of two surfaces.

We also compare inpainted color and depth maps of all types of geometry proxies. Here, for lack of space, we show results of only the *Crack* dataset as a typical case (Figure 4.10). More results are provided in the accompanying video. Note that all inpainting results have a different appearance in each trial due to the randomized initial transfer map and the different geometry proxy. Besides, none of the methods in Table 2.1 explicitly provide depth, although approaches that use *AR* markers or *SLAM* as a tracker coud provide depth from the marker position or *SLAM* points. Even though single frame appearance may be plausible, in motion, wrongly fitted planes reveal the *ROI* due to the inconsistent disparities. This effect is best observed in the accompanying video.

We specified three points on the top surface for *Single Plane*, defining an infinite plane. Consequently, the inpainted region on the side surface floats when the camera moves. *Multi-Plane* estimated two dominant planes for the top and the side surfaces. Nevertheless, this method always selects the closest plane distance from the camera; the shape of the scene resembles a concave wall-and-floor geometry. This geometry is not correctly representing the scene, leading to inconsistent inpainting results in all views except at the keyframe. In contrast, *InpaintFusion* plausibly estimates two surfaces that fit the real geometry well, resulting in seamless and consistent inpainting from various viewpoints (Figure 4.11).

4.2.3 User Study

Design We designed a repeated measures within-subjects study to compare the inpainting quality of different inpainting methods. Therefore, we introduced the indepenedent variable “inpainting method” with three conditions: *Single Plane* (S), *Multi-Plane* (M), and InpaintFusion. As dependent variables, we collected ratings for image and video results, s_{I} and s_{V}, respectively. To analyze how



Raw images with *ROI* (top) our results (bottom)

Color and normal maps of raw image data (left) and results after inpainting (right)

Figure 4.11: *InpaintFusion* in three different scenes.

Figure 4.12: Additional results. Raw images with *ROI* (left) our results (right). As we discuss in Section 4.2.3, static images do not clearly show the advantages of our method. Therefore, we strongly recommend readers to watch the results in motion in the provided supplemental video.

different geometry proxies impact inpainting results under camera motions, we also calculate the differences, $s_{\mathrm{V}} - s_{\mathrm{I}}$ scores.

Task We designed a task for rating image and video inpainting results on a 10-point Likert scale, using nine scenes including the scenes in Figure 4.11, Figure 4.12 and Figure 4.13.

Figure 4.13: Additional results. Raw images with *ROI* (left) our results (right).

Provided textual information, the participants were instructed to understand the purpose of the inpainting process in each scene, e.g., the inpainting is used to hide logos in the scene. Later we asked the participants to evaluate how well each inpainting method achieved the purpose[1]. For image results, we chose corresponding frames from each of the videos showing the different inpainting methods.

[1] For further details, we provide a supplemental video.

Apparatus We used a web-based survey system, SurveyMonkey, to collect responses from people of different expertise and nationalities, after email invitation. The participants were instructed to use at least a 13" screen to ensure reasonable viewing conditions.

Procedure After receiving textual instructions and signing an informed consent form, participants evaluated a series of inpainted images, followed by evaluation of inpainting videos. In each rating, three still images or videos were presented side-by-side, showing the original with and without the highlighted interest region and the inpainted image. 55 participants (six female, age $\bar{X}$ = 31.7, SD = 7.8 years old) volunteered for the study. On a scale from one to five, where five means the best, the mean self-rated experience concerning inpainting was 2.4 (SD= 1.1). The participants scored the nine inpainting test cases in random order. A session took approximately 17 minutes. With 55 participants, nine repetitions and three inpainting methods, we collected a total of $55 \times 9 \times 3 = 1,485$ ratings for image results and the same number of ratings for video results.

Hypotheses We did not expect significant differences in still image results where no motion disparities appear (**H1**). However, due to the fused 3D geometry proxy of the proposed method, we expected InpaintFusion to have significantly higher scores in video results than single plane and multi plane (**H2**). Moreover, we expected InpaintFusion to have significantly fewer deteriorations of the $s_V - s_I$ score than single plane and multi plane (**H3**).

Results The score data was analyzed using a non-parametric Friedman test followed by pairwise Wilcoxon signed rank tests with Bonferroni correction. The reported p-values have been Bonferroni corrected to reflect a significance level of 0.05. The statistical analysis was performed using R software.

Friedman tests revealed significant differences in image results ($\chi^2(2)$=852.02, p<0.001), in video results ($\chi^2(2)$=1018.00, p<0.001), and in $s_V - s_I$ scores ($\chi^2(2)$=1018.00, p<0.001). Figure 4.14 summarizes the study results. For Post-hoc test results refer to Table 4.1.

Discussion InpaintFusion is scored one unit higher in the median than the others in image results. Therefore, we reject our pessimistic hypothesis H1, as InpaintFusion performed better than expected. One possible explanation could be that the geometry term of InpaintFusion constrains the search range to a proper region, while single plane must search pixels only using colors. We observed that multiplane tends to leak colors in wrong plane regions, when it fails to estimate planes correctly.

For video, InpaintFusion scored even one unit higher than for still images, clearly outperforming

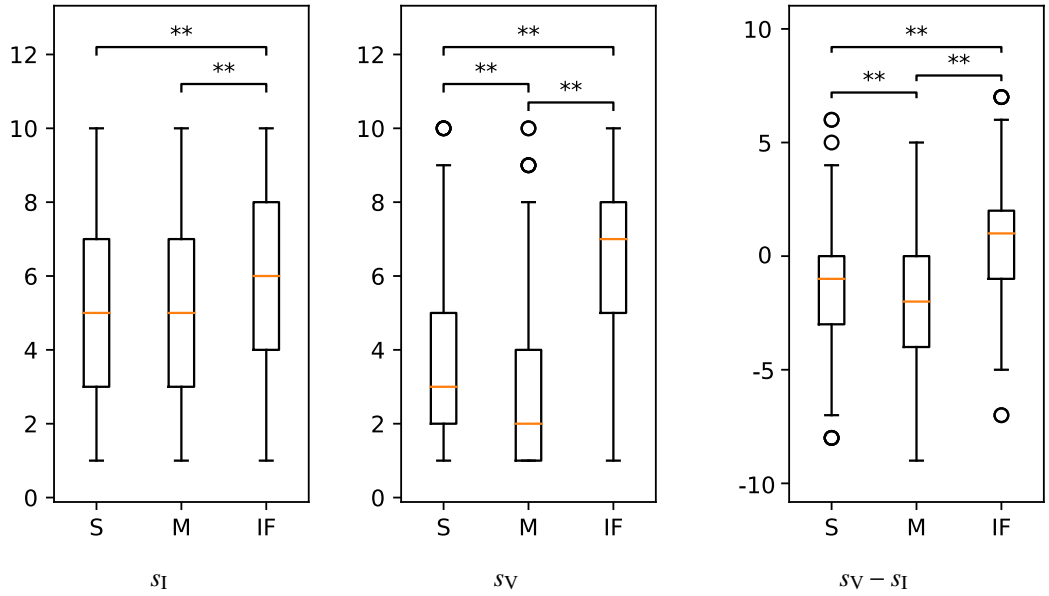


Figure 4.14: Study results in image evaluation (a), video evaluation (b), and deteriorations from image to video (c). Looking at the median values (orange lines), we can conclude that InpaintFusion outperforms Single Plane and Multi-Plane approaches.

both Single Plane and Multi-Plane. We explain this by the well-maintained temporal and spatial coherence of InpaintFusion under 6DOF motions. InpaintFusion gave the participants better impressions than in image results, while scores for Single Plane and Multi-Plane were lowered due to geometrical misalignments. Single plane repeatedly failed when there were multiple objects of interest at different depths or when the region had varying depth. Also, we observed that, when multiplane fails to estimate the right planes, mismatched disparities in the video are created. Figure 4.10 shows such typical cases. Overall, we accept our optimistic hypotheses H2 and H3. We conclude that InpaintFusion can maintain the quality even in scenes where planar inpainting approaches fail.

4.2.4 Runtime Performance

We implemented *InpaintFusion* on a notebook computer (Intel Core i7-6567U with 3.3 GHz, 16 GB RAM, external NVIDIA GeForce GTX1080Ti connected via Thunderbolt 3) running Windows 10. As RGB-D sensor, we either used a Microsoft Kinect v1 or an Intel RealSense SR300, running at 30 Hz in 640×480 resolution. We implemented our system in two threads, one performing keyframe inpainting and another one for the rest of the processing, including Graphics Processing Unit (GPU) tasks (using OpenGL and GLSL), *SLAM*, rendering, and passing selected keyframes to

Table 4.1: Post-hoc tests, methods in rows compared against methods in columns. Post-hoc tests indicated that InpaintFusion scores (Mdn=6) were statistically higher than Single Plane scores and Multi-Plane scores in image results. Results indicated that all combinations in video results have significant differences; InpaintFusion scores (Med=7) were statistically higher than Single Plane scores and Multi-Plane scores, and Single Plane scores were statistically higher than Multi-Plane scores. Also, in $s_V - s_I$ scores, post-hoc tests indicated that all combinations have significant differences; InpaintFusion scores (Med=1) were statistically higher than Single Plane scores and Multi-Plane scores, and Single Plane scores were statistically higher than Multi-Plane scores.

		Single Plane	Multi-Plane
Single Plane	S_V	-	Z=-7.342 p<0.001 r=0.330
	$S_V - S_I$	-	Z=-6.366 p<0.001 r=0.286
InpaintFusion	S_I	Mdn=5 Z=-5.540 p<0.001 r=0.249	Mdn=5 Z=-4.66 p<0.001 r=0.209
	S_V	Mdn=3 Z=-16.589 p<0.001 r=0.746	Mdn=2 Z=-17.775 p<0.001 r=0.799
	$S_V - S_I$	Mdn=−1 Z=-15.071 p<0.001 r=0.677	Mdn=−2 Z=-16.251 p<0.001 r=0.730

the inpainting thread.

Table 4.2 and 4.3 summarize the performance in milliseconds in the main and inpainting threads, respectively. Overall, *InpaintFusion* operates approximately at 31.16 Hz. Although the first keyframe inpainting is most expensive and finishes in 4.4 sec., it runs in the background without interfering with tracking, and it takes less in the following keyframes due to keyframe propagation. In comparison with existing methods using a similar hardware setup, *InpaintFusion* performs equal or even faster, even though it is capable of full 3D inpainting that has never been achieved before (please see the accompanying video).

Table 4.2: Average time (ms) spent in each stage of the main thread.

Component	Runtime [ms]
Tracking (OpenCV / GPU ICP)	128.16 / 21.44
SLAM fusion	4.60
Inpainted KF fusion	8.00 / KF
View-dependent KF blending	1.93
Misc (Frame pre-processing & data handling)	3.20
Total (OpenCV / GPU ICP)	137.88 / 31.16

Table 4.3: Average time spent in each stage of the inpainting thread, which runs in parallel to the main thread and therefore does not stall the application.

Component	Runtime [ms]
KF propagation	19.75
Transformation map optimization (50 raster-scans at each of a six level pyramid)	4288.33
Depth estimation from depth samples	96.67
Mask ratio	17.02 %
Total	4385.00

4.3 Limitations and Future Work

While *InpaintFusion* generalizes the inpainting-based *DR* methods, some points need to be addressed to further improve the quality.

Integer space transformation map propagation The transformation map f represents pixel-to-pixel offsets, i.e., $f \in \mathbb{Z}^2$. Therefore, propagating such a map to the next keyframe leads to nearest-neighbor interpolation in image space. Such an approach is prone to aliasing, as seen in $C_k(f_k^*(\mathbf{u}))$ of Figure 4.5 (e). A potential solution to this problem in our system is to set fairly large thresholds discussed in Section 4.1.5. We could re-optimize the propagated transformation map using the warped color map from the closest keyframe as a constraint with higher weight in the appearance costs minimization [50]. However, once the optimization finds better pixel-to-pixel relationships than the current ones, the resulting appearance of the inpainting will differ substantially from the other keyframes. Planar inpainting avoids this problem, since it does not have to handle occlusions. To mitigate the aliasing problems, we can use view-dependent keyframe blending, as described in Section 4.1.9, to compose multiple keyframes to render the current inpainted region.

Occlusion handling with real and inpainted depths In case our system reconstructs a potentially observable background, and the user perfectly labels an object of interest before inpainting starts, our system can safely project the reconstructed surfels belonging to L_O to exclude ones with L_{ROI} from inpainting. However, in practice, the surfel-wise colors are not precise enough to fill in L_{ROI} as they are. This issue is demonstrated in an existing attempt [98], and surfels with view-dependent properties could mitigate this problem [104]. Therefore, we chose to inpaint the entire *ROI* in the first keyframe. This means that, in case a real background is observed after inpainting, that inpainted depth and real depth may disagree, leading to discontinuities at the *ROI* border. To minimize such discontinuities, we can use automatic segmentation, using the manually specified *ROI* as seed, to obtain a better labeling that supresses problems at borders.

Bundle inpainting using all keyframes Adding more constraints in the transformation map optimization will lead to more robust inpainting, but it will also further restrict the pixel search range. This can lead to a lack of pixel sources. One could search pixels in all preserved keyframes to optimize a single transformation map, but this would require another term in the optimization to represent pixel continuities across keyframes. One good example we could find projects multi-view images to common planes to use available pixel sources for the inpainting [108], although the authors stress the difficulties to apply the strategy for non-planar regions. We find such an extension an interesting avenue of future research.

4.4 Summary

We presented a novel approach for interactive image inpainting in 3D. We have shown how the integration of fusion and multi-keyframe inpainting delivers globally consistent and appealing results. Our system ensures frame-to-frame coherence of the inpainted results by considering a 3D geometric term in addition to texture in image space. This ability improves the range of possible use cases for interactive *DR* applications, for instance, we can target multi-view rendering for stereoscopic display devices. Furthermore, our system supports image editing by its ability to add 3D visual effects to inpainted images. This enables quickly adding 3D visual effects to images and videos, providing a tool for previewing image and video editing operations.

InpaintFusion also opens up possibilities for *AR* effects after the inpainting. For example, we demonstrated relighting from virtual car headlights and physical animation of snowballs in the inpainted region (Figure 4.1). We also made real objects virtually interactive by replacing the real object with a scanned model after removing the real object. In case the user could scan the

[2] https://www.gcc.tu-darmstadt.de/home/proj/mve/

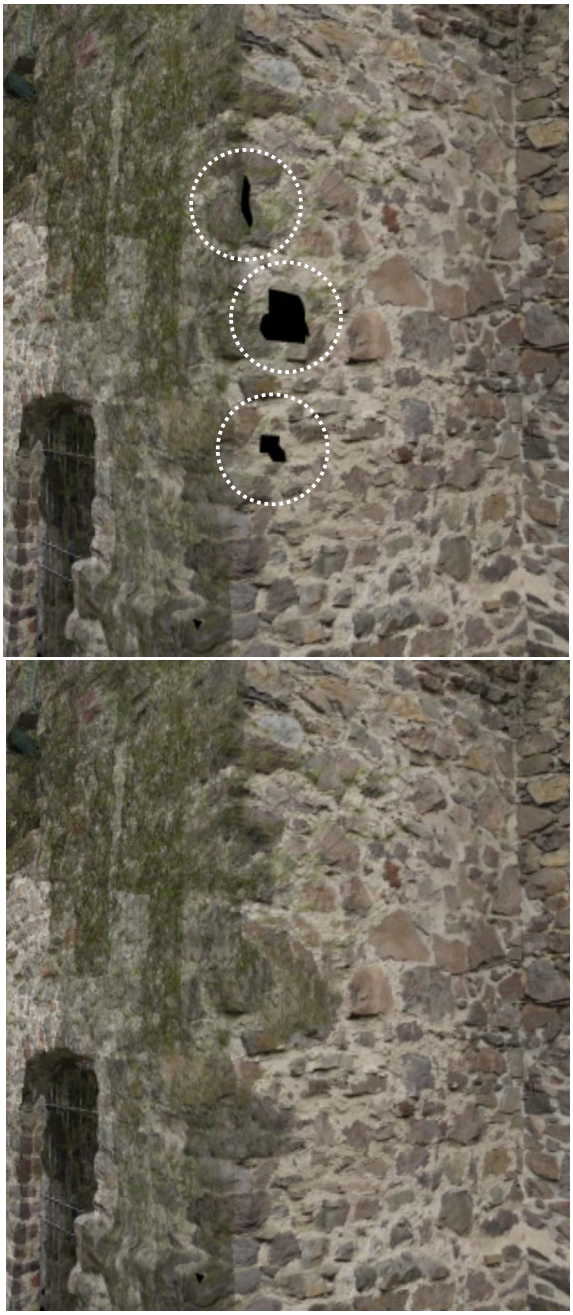


Figure 4.15: Scene completion using *InpaintFusion*. Our approach is able to complete 3D reconstructions. In this example, it completes CityWall dataset provided by TU Darmstadt[2] and rendered by our *IBMR* pipeline. *InpaintFusion* generates the missing geometry and color information (right), which is highlighted by white circles in the image on the (left).

backgrounds in advance, our system can erase frontally occluding objects for X-ray vision. In this case, surfels belong to the *ROI* are replaced with observed ones. Most importantly, *InpaintFusion* supplies RGB-D inpainting for scene completion, filling holes in a point cloud corresponding to unobserved areas in the scene or reconstruction failures, which has a crucial role in tele-operation task. (Figure 4.15). In addition to scene completion, inpainting further improves the tele-exploration experience by removing unwanted objects (see Figure 4.17) or graffiti (see Figure 4.16) from the scene. In Chapter 5, we will present how interactive remote exploration scenarios can benefit from *IBMR*.

Figure 4.16: Removing unwanted graffiti on the well from *IBMR* result. Noticeable misalignment of the tile grid is the result of a trade off between spatial similarity versus texture similarity. While increased spatial similarity constraint would alleviate the problem by copying group of pixel together from source, reduced texture similarity can cause a visual difference between inpainted region versus the rest of the image.

Figure 4.17: Removing a historical cannon ball from a city wall, rendered using our *IBMR* pipeline.

5 Drone-Augmented Human Vision

Contents

In this chapter we demonstrate that the use of Image-Based Modeling and Rendering (IBMR) also facilitates tele-exploration scenarios where physical interaction with the remote enviroment is needed (Figure 5.1). Especially, we demonstrate that rendering from an exocentric point of view with the flexibility to change viewpoint reduces cognitive load of the user. In addition, our flexible viewpoint rendering gives the user the ability to move in the scene virtually as desired.

To better show this impact, we used an indoor camera drone and streamed its images to an *IBMR* pipeline. An interactive and highly mobile application scenario which lets the user move physically, is enabled by using a mixed-reality device, namely the HoloLens from Microsoft (see Figure 5.2). Due to the performance limitations of the device, in contrast to a full *IBMR* pipeline, we only used the most recent image streamed from the drone's camera in our visualization to introduce novel interaction techniques to tele-exploration tasks.

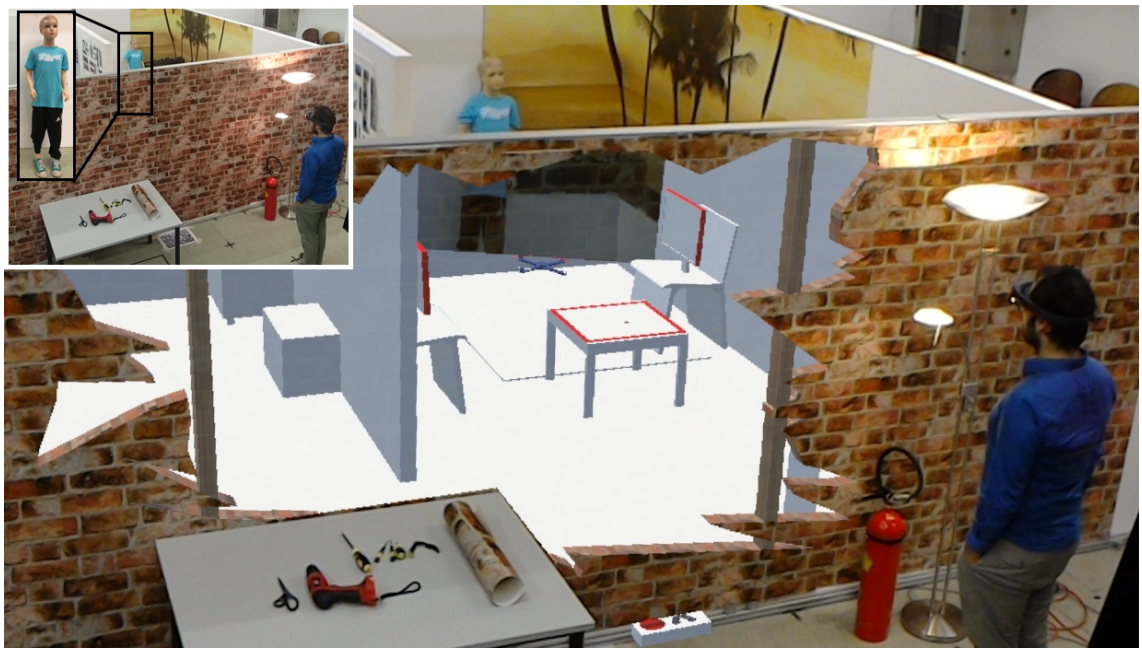


Figure 5.1: This image was captured live with a second HoloLens on a tripod. The drone-augmented human is interacting with the drone in an exocentric view. The human is steering the camera drone via gaze direction and perceives an X-ray-like vision into occluded areas. The brick pattern is a physical wallpaper. The lower part of the mannequin is applied as a perspectively correct texture and extends the user's perception of the visible upper part of the mannequin.

5.1 Interface Design

Our drone-augmented human vision system lets the pilot control a drone inside an occluded space indirectly, via an exocentric visualization provided in a see-through Head-Mounted Display (HMD) (Microsoft HoloLens). While the drone travels in the remote environment, the video frames streamed from the on-board camera are projectively texture-mapped onto a geometric model of the scene. The scene is rendered from the user's current perspective, as measured by the built-in self-localization of the *HMD* (see Figure 5.3).

In addition, a virtual representation of the drone is rendered at the position reported by the physical drone, to give the pilot an overview of the physical configuration of the occluded space. The interior scene with partial texture mapping is made to appear inside a "cutaway" magic lens that appears as a hole in the occluding wall structure.

For flight control and navigation in the occluded space without hitting obstacles, we introduce two interaction techniques, called *pick-and-place* and *gaze-to-see*. Moreover, we introduce *overview-*

Figure 5.2: HoloLens. © Microsoft

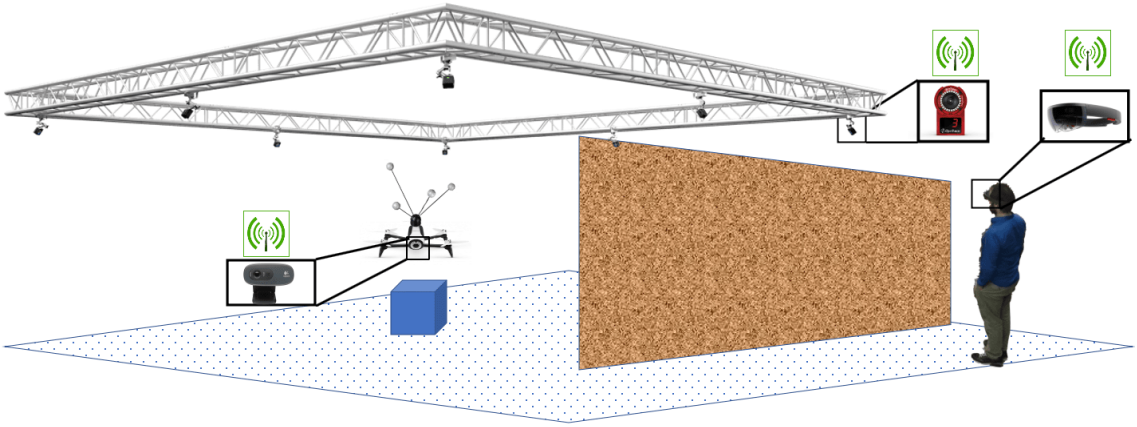


Figure 5.3: System set up overview. A user with a *HMD* looks through a wall to see the occluded object of interest (blue cube). Using the Optitrack motion tracking system, mounted to the metal frame, drone's pose is calculated and streamed to *HMD* together with the images obtained from on-board drone camera. Entire system communicates through WiFi network.

and-detail, a transitional interface [7] to reveal details on demand.

5.1.1 Pick-and-Place

This interaction technique allows users to pick a drone by looking at it and applying a pinch gesture. After picking the drone, moving one's hand repositions the drone in 3D space, as illustrated in

Figure 5.4. The hand movement is scaled proportionally to the distance of the picked object, as in the scaled-world-grab technique proposed by Mine et al. [92]. More formally, the displacement vector $\vec{D}_{drone}$ of the drone's position P_{drone} in $\mathbb{R}^3$ is calculated as

$$\vec{D}_{drone} = \frac{\left\|P_{drone} - P_{eye}\right\|}{\left\|P_{hand} - P_{eye}\right\|} \cdot \vec{D}_{hand} \tag{5.1}$$

where P_{eye} and P_{hand} represent the positions of the eye and the hand, respectively, while $\vec{D}_{hand}$ indicates the hand's motion vector in $\mathbb{R}^3$ (Figure 5.4). P_{eye} and P_{hand} are directly provided by HoloLens, whereas P_{drone} is received from the drone tracking system. Note that, depending on factors like dominant eye, *HMD* position on the head or the distance of the currently focused object, P_{eye} may be subject to brittle calibration. However, during our experiments, users did not indicate that they needed (re-)calibration.

5.1.2 Gaze-to-See

Using the view vector and eye position provided by HoloLens, one can calculate the point of interest P_{gaze} a user is gazing at by intersecting the viewing ray with the scene model. Knowing gaze position allows to predict which part of the occluded scene a user is interested in. Therefore, in this interaction technique, the drone focuses on the high level goal of the user and automatically repositions to observe the area around the user's point of interest with its on-board camera. Let $\vec{N}_g$ be the normal vector at P_{gaze}, and let $\vec{Z} = \{0, 0, 1\}$ denote the up-axis of the scene. The drone is positioned at

$$P_{drone} = P_{gaze} + \frac{\vec{N}_g - (\vec{N}_g \cdot \vec{Z}) \cdot \vec{Z}}{\left\|(\vec{N}_g - (\vec{N}_g \cdot \vec{Z}) \cdot \vec{Z})\right\|} \cdot x \quad \text{if} \left\|\vec{N}_g \cdot \vec{Z}\right\| < \left\|\vec{N}_g\right\| \cdot 0.9 \tag{5.2}$$

Unless we are looking at a horizontal surface, the drone will reposition x meters away from the point of interest along a displacement vector corresponding to the surface normal projected to a horizontal plane (Figure 5.5). If the user is looking at a horizantal surface then no target position will be set for the drone.

In our experiments, we set $x = 0.5$ meters for ensuring a close-up view of the surface. The drone's yaw orientation is adjusted to align with the negative displacement vector. In case the user looks at a horizontal surface, the drone is positioned between the user and the point of interest, mimicking the user's view vector in the horizontal plane. If the calculated position is not inside the safe flight zone, the repositioning terminates at the nearest border of the permitted flight zone.

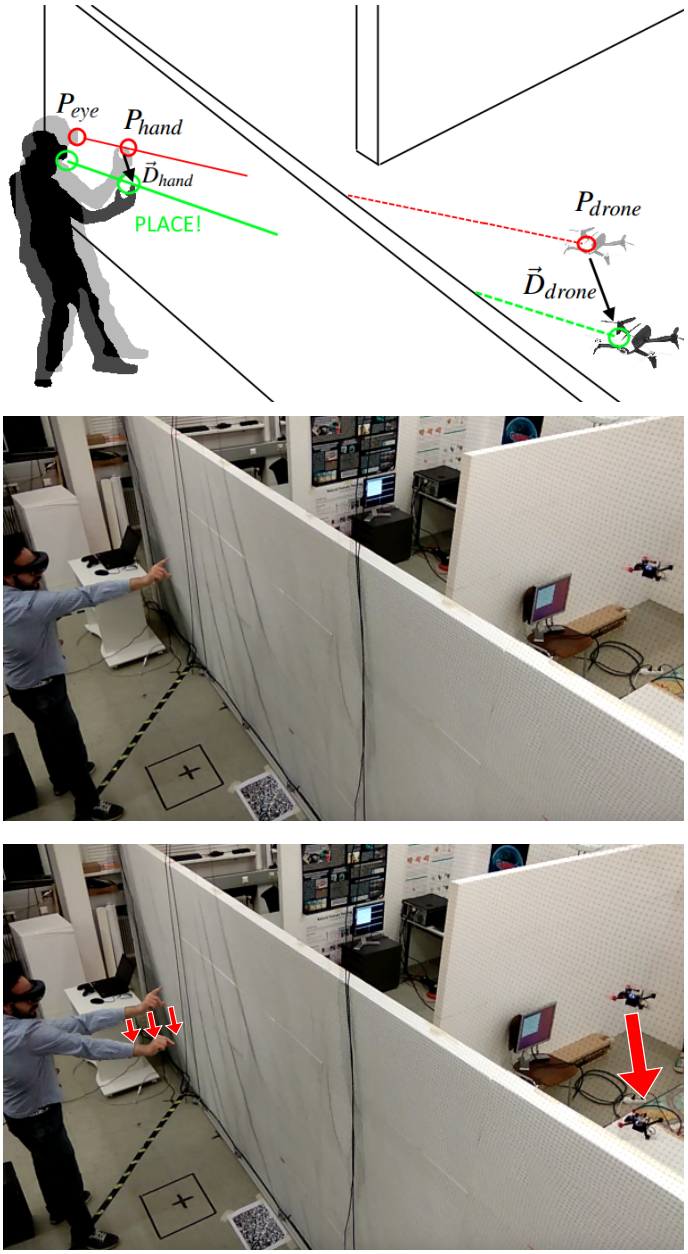


Figure 5.4: Pick-and-place – the user picks and places the drone, as if the drone is at the reach of the user's arm.

5.1.3 Overview-and-Detail

By visualizing the occluded scene and the drone from a user's perspective, our system allows a drone pilot to better understand the spatial relationships between scene geometry, drone and the

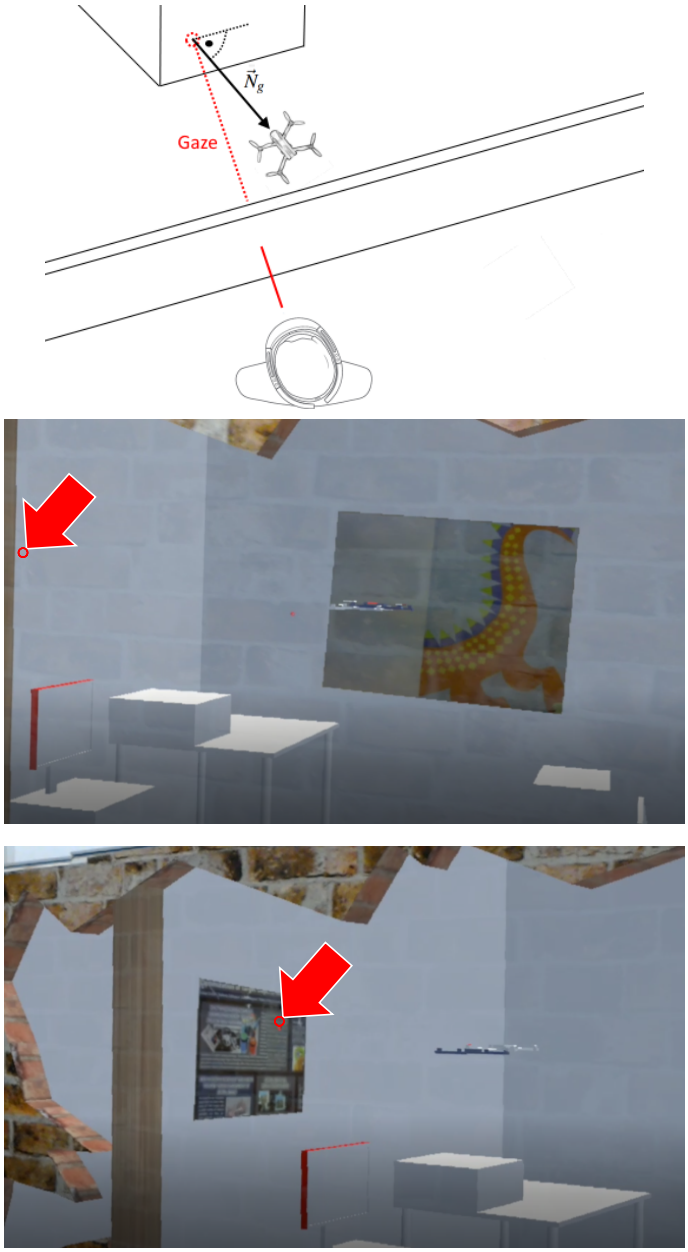


Figure 5.5: Gaze-to-see – the user looks at a point as shown by the red arrows and steers the drone to a position where the drone observes the point of interest and ensures a close up view.

pilot's body. However, this visualization lacks details, as the drone can be far away and both camera and display suffer from a rather limited field of view. Therefore, we introduced an overview-and-detail technique, which fills the gap between egocentric and exocentric drone control modes in the form of a transitional interface [7] using image-based warping [127]. After steering the drone to a

point of interest, users are given the option to either virtually move closer to the drone or to the currently gazed-at surface point in the occluded scene, by selecting the corresponding interface hotspot. During the detail visualization, we apply the occluding wall structure to clip the zoomed detail geometry in order to avoid confusion between real and occluded virtual geometry (Figure 5.6). Zooming in is achieved by positioning the virtual hole in front of the gazed point while preserving the relative transformation between the virtual hole and the user's camera view. The position of the virtual hole is computed in the same way as the positioning of the drone in gaze-to-see interaction.

5.2 Implementation

A detailed overview of our experimental system architecture, including hardware and software components and data flow between them, is shown in Figure 5.7. The system builds on the drone design described by Isop et al. [63] and is based on six main components:

We use an (1) Optitrack motion tracking system consisting of a server system with 12 cameras to externally localize the drone. The Optitrack is connected to (2) a ground-station, which further communicates to (3) the drone's on-board computer, an Odroid XU3, and (4) the HoloLens via WiFi. For our user study, we complemented the system with a remote control user interface including (5) a joypad for steering and (6) a visualization station. All components communicate via Ethernet or WiFi.

The software components are integrated via Robot Operating System (ROS) [109] nodes. We use Unity 3D for visualization on the HoloLens and the *ROS* tool *RViz* for monitoring on the ground-station.

The motion capturing node on the ground-station relays User Datagram Protocol (UDP) packages from the Optitrack system, which describe timestamped poses of the tracked objects, to the Odroid. The Odroid transforms the poses into local coordinates of the drone and forwards them to the MAVROS node, a *ROS* wrapper to communicate with the Pixfalcon autopilot via publish/subscribe messages. It is responsible for acquiring Inertial Measurement Unit (IMU) data, pose updates, target coordinates (setpoints), internal pose estimates, etc.

The drone controller node on the HoloLens maps gestures into target drone position and visualizes the drone's current position and target positions in the mixed-reality view. Setpoints, target values for the drone positions, can be generated either by the HoloLens interface or by the joypad interface.

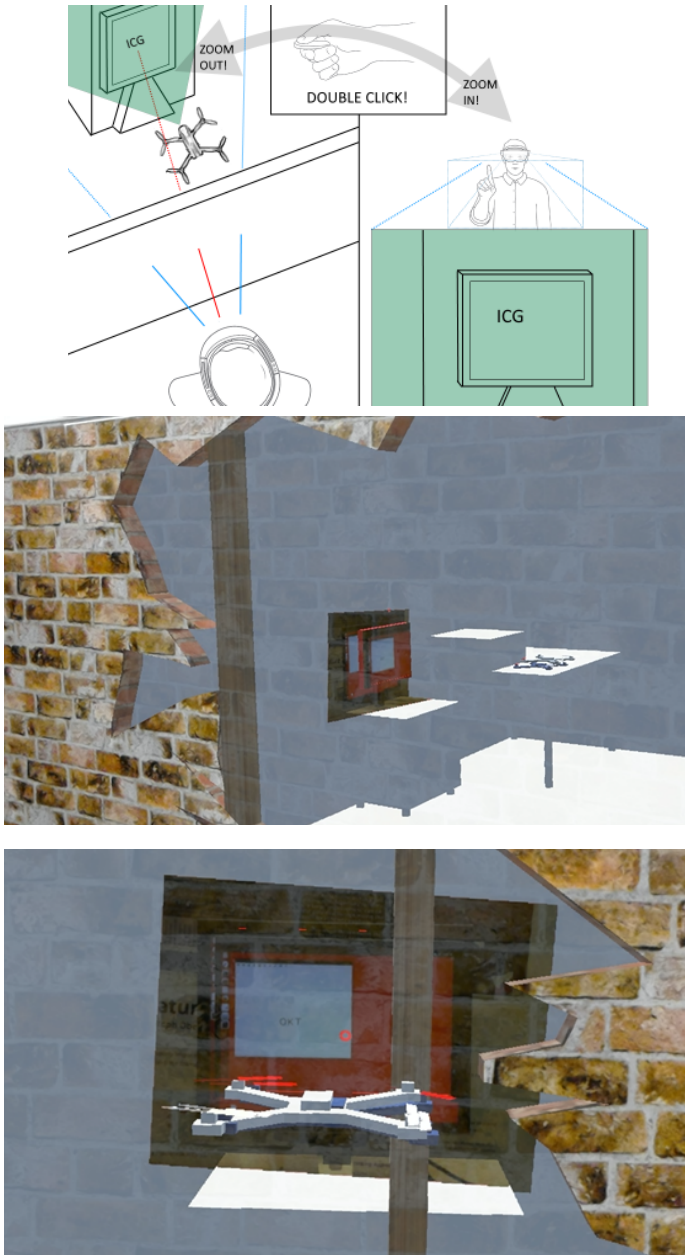


Figure 5.6: Overview-and-detail – when a far point in the scene is investigated, the user can virtually fly close to the point of interest and have a virtual viewpoint in the scene. However, all virtual scene elements remain behind the wall to avoid confusing depth perception.

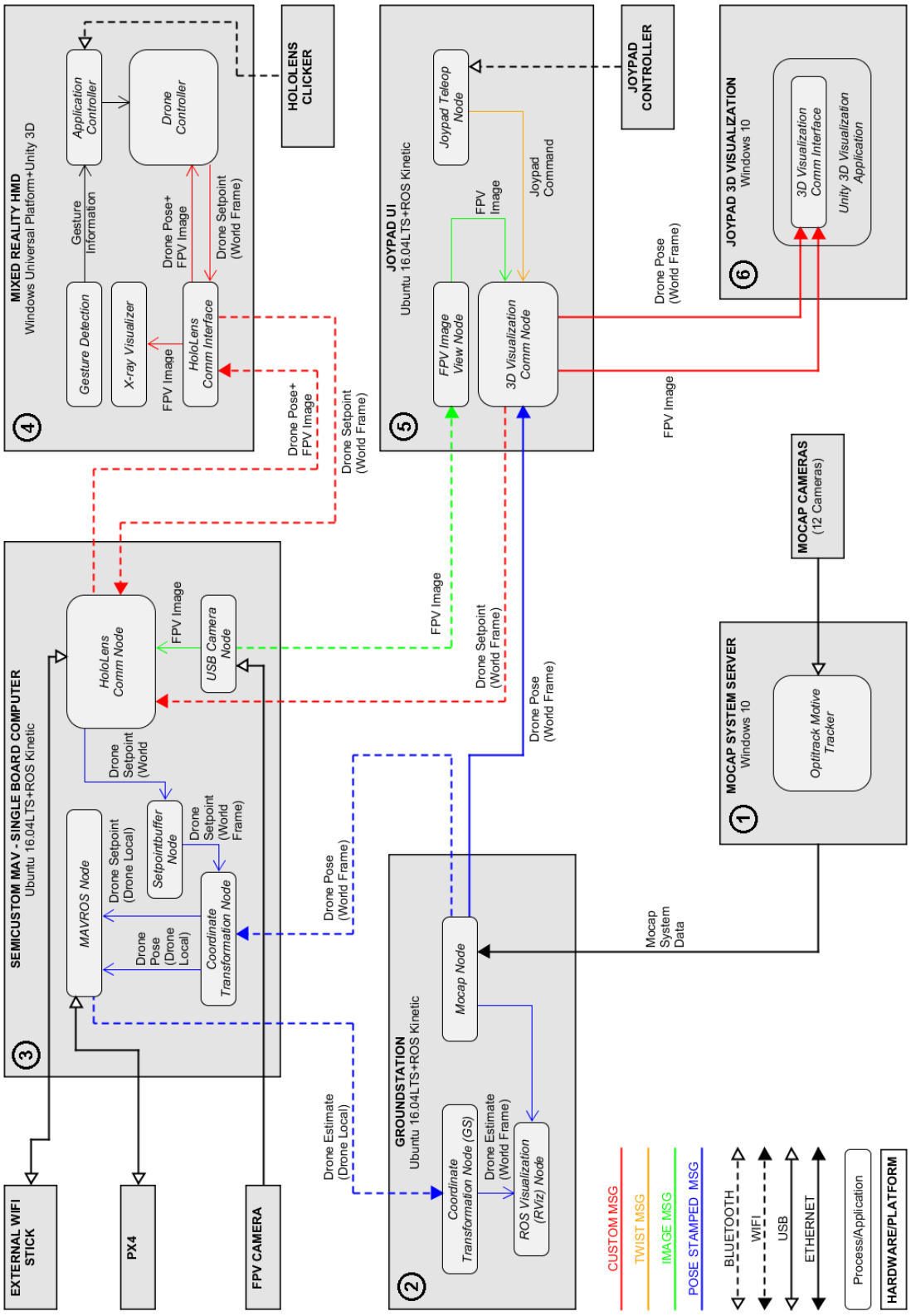


Figure 5.7: Overview of main hardware and software components of our experimental setup including (1) the motion tracking system, (2) a ground station, (3) the on-board computer of our semi-customized drone, (4) the HoloLens, (5) a joypad interface, and (6) a visualization station.

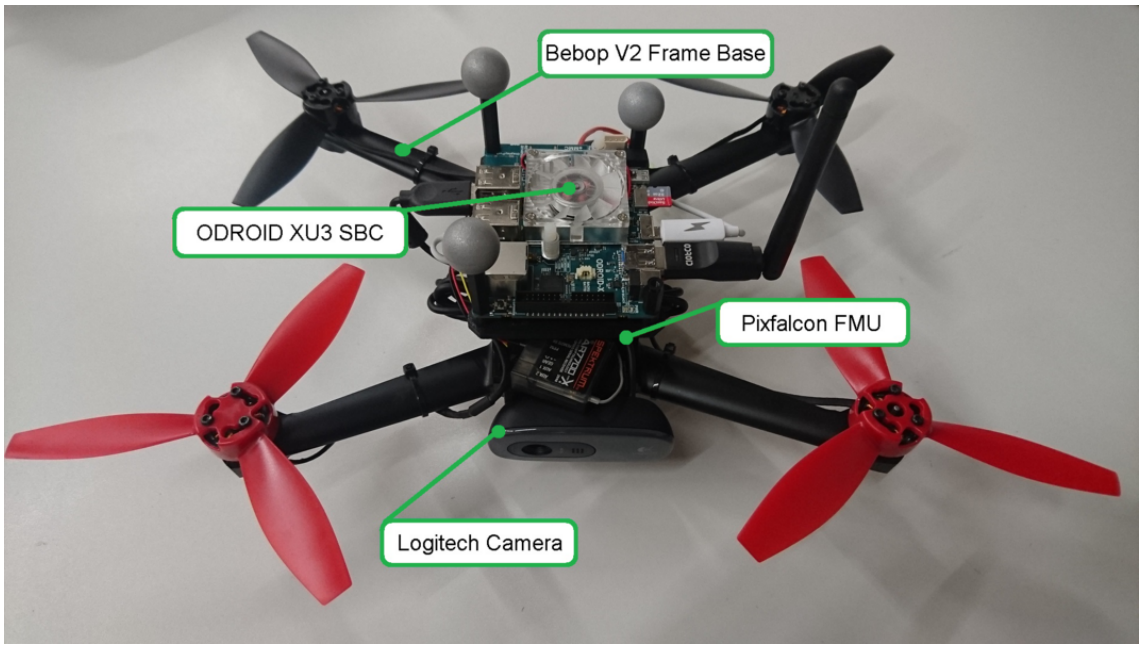


Figure 5.8: Experimental drone setup including the main components. The camera (Logitech Camera) is mounted facing forward and captures the images needed for visualization of occluded space. Inside the frame, the autopilot (Pixfalcon FMU) is located which ensures drone reaches a given position. The battery is mounted on the bottom to balance weight distribution. The onboard computer (ODROID XU3 SBC) is located on top and communicates with the ground station and do the necessary commanding of drone.

5.2.1 Drone Setup

The drone (Figure 5.8), which has a frame with 25 cm diameter and weighs 450 g, uses a semi-customized design with rotors and frame taken from a Parrot Bebop 2 platform. The flight time is about 11-15 minutes, while running all relevant components and tasks. We added a PX4 Pixfalcon autopilot as a low-level flight control unit and an Odroid XU3 single-board processor computer.

The forward-looking camera captures image data at 30 Hz with 640×480 resolution and delivers it to the Odroid via USB. The video is streamed to the HoloLens in MJPEG format, annotated with timestamp and camera poses to allow precise image-based rendering. All high-level tasks, including processing of image data, estimated poses from the motion tracker and control commands received from the pilot run on-board and are implemented in the *ROS* framework.

5.2.2 Flight Management Control

For localization of the drone, we use the external Optitrack system with 12 ceiling-mounted tracking cameras, covering an area of roughly $5 \times 4 \times 3m$. The Optitrack system provides pose estimations at 120 Hz, which are delivered over WiFi to the Odroid at a latency of ~25 ms. The serial link from the Odroid to the Pixfalcon autopilot adds another ~10 ms of delay. The system time between Optitrack server, ground-station and Pixfalcon autopilot is synchronized based on Network Time Protocol (NTP) using the Chrony service.

Designing a drone for autopilot-controlled flight at low heights in small confined spaces is challenging, because of the imminent danger of hitting obstacles. We combined several measures to ensure safe operation. Since high flight speed was not a primary goal, we used low-thrust engines (taken from a Bebob 1 platform) and soft materials for the propellers. This produces less turbulences when flying close to walls and around objects. We further relied on the ability of the Pixfalcon autopilot to use pure inertial navigation for short periods, when the measurements from the motion capture system are noisy or intermittent. Such a noise can arise from the ambient lighting changes and reflections emitted from metallic surfaces. The pose updates are buffered on the Odroid to minimize the occasions where the Pixfalcon autopilot switches unintendedly from autonomous flight mode into manual mode if the Wifi link stalls or drops position updates from the Optitrack.

5.2.3 Control of Drone Movements

Control of the drone is based on measuring its Six Degree of Freedom (6DoF) pose by the motion capture system in world coordinate representation. We make use of the Pixfalcon autopilot inertial estimator to fuse the motion capture data with the inertial sensors of the Pixfalcon autopilot, deriving 3D position $[x, y, z]$ and the yaw θ required for the drones's position control. These measurements, obtained at discrete times $i = 0 \ldots n$, are denoted as Y_i.

For position control, we use the internal linear control approaches of the Pixfalcon autopilot. The methods consist of an inner attitude rate PID (proportional/integral/derivative) controller with pitch, roll and yaw angular velocities as inputs. This control loop is enclosed by an attitude P-controller with attitude setpoints for roll, pitch and yaw angles and throttle as reference input. The inner control loop is nested in a position control loop, which takes 3D position $[x, y, z]$ and yaw θ as reference inputs H_i, which can, for example, be derived from the HoloLens interaction. The yaw

reference is directly fed into the inner attitude control loop.

$$Y_i = \{x_i, y_i, z_i, \theta_i\} \quad (i = 0...n) \tag{5.3}$$

$$H_i = \{x_i, y_i, z_i, \theta_i\} \quad (i = 0...n) \tag{5.4}$$

$$E_i = H_i - Y_i \tag{5.5}$$

The derived position error, given in Equation 5.5, is calculated in every iteration i and fed into the control structure of the Pixfalcon autopilot. We use aggressive controller gains, which are based on the default gains of the more heavyweight DJI F330 model, to establish fast response times and accept slight overshooting of approximately 5%, when the drone's actual position converges towards the given setpoint.

5.2.4 Precomputed Path Planning

For our experiments, we wanted to relieve the pilot as much as possible from path planning, providing the illusion of augmented vision without concerns about flight safety. However, fully featured path planning is computationally expensive and can be brittle. Since we track the drone externally, rather than by Simultaneous Localization and Mapping (SLAM), we can pre-compute the necessary path planning information from the floor plan. In our test environment, we divided the space into three regions: two rooms connected by a corridor.

If the pilot issues a repositioning command that requires changing the region, the path planning first approaches a predefined waypoint at the boundary before progressing to the neighboring region. Overall, our path planning is simplistic, but works instantaneously and reliably prevents accidents due to hitting obstacles or walls of the scene. A more realistic path planning based on *SLAM* would run an A* algorithm on a map of the environment that has already been explored by the drone.

5.2.5 Joypad Control

Alternatively to the path planning, the drone can be controlled via a joypad. In this case, a custom *ROS* node integrates the inputs from four axes of the joypad and converts them into a 3D position and yaw of the drone. We derive the position reference commands by integration of the joypad's linear axis commands J_i over the time intervals between discrete times i. The position error E_i in this case is given as $E_i = J_i - Y_i$.

To enable a fair comparison between the exocentric interaction techniques introduced in Section 3 and the joypad interface, we added advanced features to the joypad interface, which go beyond

what is conventionally available in commercial drone control.

First, we provide drift-free stabilization of the Micro Aerial Vehicle (MAV) position during navigation in the scene. This kind of stabilization is not available when using off-the-shelf drone technology. Conventional tracking and stabilization, especially in the x-y plane, is usually based on opticalflow or inertial sensors, which suffer from drift over time. With the drift-free tracking, we also enable a basic level of disturbance rejection against turbulences which occur during flight in narrow parts of the scene.

Second, we chose Mode-2 axis mapping on the joypad, which is a well-known and widely accepted mapping for drone control. It is also the default configuration in a variety of off-the-shelf drone products, e.g., the Parrot AR Drone 2.0, the Parrot Bebop 1/2, and the DJI Marvic. Mode-2 mapping employs the left joystick for commanding vertical velocity and velocity around the rotational z-axis of the drone. No direct thrust control is required by the user, reducing cognitive load. The right joystick controls the translational velocity in X and Y direction.

Third, we created a safe-guard for the use by introducing artificial boundaries inside the scene, so the user is not able to crash the drone into walls or hit obstacles. Before each experiment, the user was informed that crashing the *MAV* is not possible. We presented visual feedback when the user hits the artificial boundaries via warning message, and we visualized the valid flight areas inside a 3D perspective view with green bounding boxes (Figure 5.9). If the user hits the boundaries, the drone did not fully stop, but continued movement along the boundary with the resulting speed vector. Thus, the user was able to "slide along" the artificial boundaries. Another safety mechanism allowed the joypad user a simple and safe transition between the rooms. Once the user approached the narrow corridor between the rooms, the drone was automatically transported to the other room. We did not impose any limit in z-direction, so the user was able to safely transit between the rooms at any flight height.

5.2.6 Head-mounted Display

The pilot interface runs on the HoloLens. Its tinted visor holds transparent combiner lenses, in which projected images are shown to the user. We rely on the built-in *SLAM* system of the HoloLens to provide continuous self-localization. In order to register the localization data reported by the HoloLens with the Optitrack coordinates (OC), we use a Vuforia tracking target. The tracking target is placed on the floor in front of the occluding wall, which corresponds to the plane $Z = 0$ in OC. The transformation between OC and tracking target was calibrated offline. Using the Vuforia SDK for HoloLens, we obtained the transformation from the origin of the HoloLens *SLAM* tracking

to the tracking target at startup time and concatenated to the OC transformation. Thus, a drone pose reported in OC can transformed into HoloLens coordinates.

5.2.7 X-ray Vision

We apply Augmented Reality (AR) X-ray vision while providing the user with an exocentric interface for nearby remote scenes. We use the Unity 3D game engine for rendering the scene geometry on the HoloLens. A stencil masking technique is applied to render X-ray visualization only where the virtual geometry is observed through the virtual hole in the wall.

Images for first-person view are streamed from the drone-mounted camera as MJPEG, annotated with the drone's pose when the frame was taken. The MJPEG is decoded and uploaded as a texture to the Graphics Processing Unit (GPU) of the HoloLens to generate the Mixed Reality (MR) view.

For each fragment displayed on the HoloLens, the texture is sampled during the shading process by projecting fragment positions in world space with the view projection matrix of the drone's camera. To eliminate virtual geometry from being rendered between the occluding wall and the user, fragments with world coordinates that are located behind the wall plane are discarded.

5.3 User Study

We conducted two user studies to collect quantitative and qualitative data on the performance and scalability of our system.

5.3.1 Physical Viewpoint Study

First, we were interested in users' spatial awareness using the exocentric viewing interface and X-ray vision, compared to a standard egocentric interface that lets the pilot control the drone with a joypad. Specifically, we studied the case in which the user is in-place investigating the occluded scene, which is close (e.g. behind a wall), but cannot be reached from the current viewpoint. To ensure a fair comparison, we supported the joypad user not only with the live egocentric video from the drone, but also with a screen-based 3D visualization of the hidden space, showing real-time updates of the drone's position. We formulated our hypotheses as follows:

H_1: *"Steering a drone for collecting information in distant spaces is faster with the exocentric interface than using a common joypad interface."*

H_2: *"Steering a drone for positioning in distant 3D spaces is faster with the exocentric interface than using a common joypad interface."*

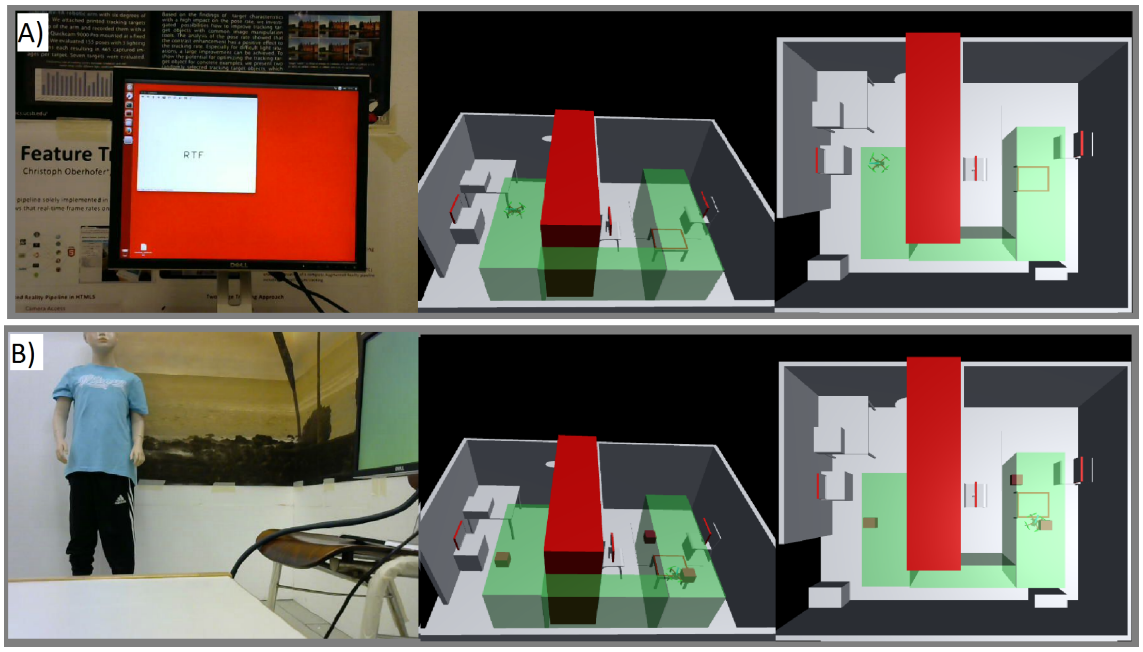


Figure 5.9: The first-person view and two additional views of the flight space were available for EGO user during both tasks. The red zone in the middle of the 3D model indicates a restricted flight zone, where the drone's position is confined to remain inside boundary, while the green zone delimits the allowed flight space. (a) The user's view while engaged in the screen-reading task. The model shows three screens with red borders, but only two of them are active per user. (b) The user's views during the drone positioning task. Cubes in the 3D model indicate the target positions to be reached by the drone with a tolerance of 10 cm.

H_3: *"Steering a drone for collecting information and positioning in distant 3D spaces is more intuitive with the exocentric interface than using a common joypad interface."*

Study design and tasks To test our hypotheses, we chose the interaction mode as an independent variable with two conditions: Exocentric interface (EXO) and egocentric interface (EGO). In addition, we selected completion time as a dependent variable. Workload was measured using the NASA Task Load Index (TLX) [45], and overall preferences of the users were assessed via semi-structured interviews. Based on a within-subjects design, participants were given two instances of a search-and-explore task to be accomplished with either of the interaction methods, in randomized order.

Reading text on monitors We asked subjects to steer the drone with both interfaces and report random texts displayed on two monitors positioned in different places of the occluded space. The monitors showed different background colors (red and green) to uniquely identify them from an

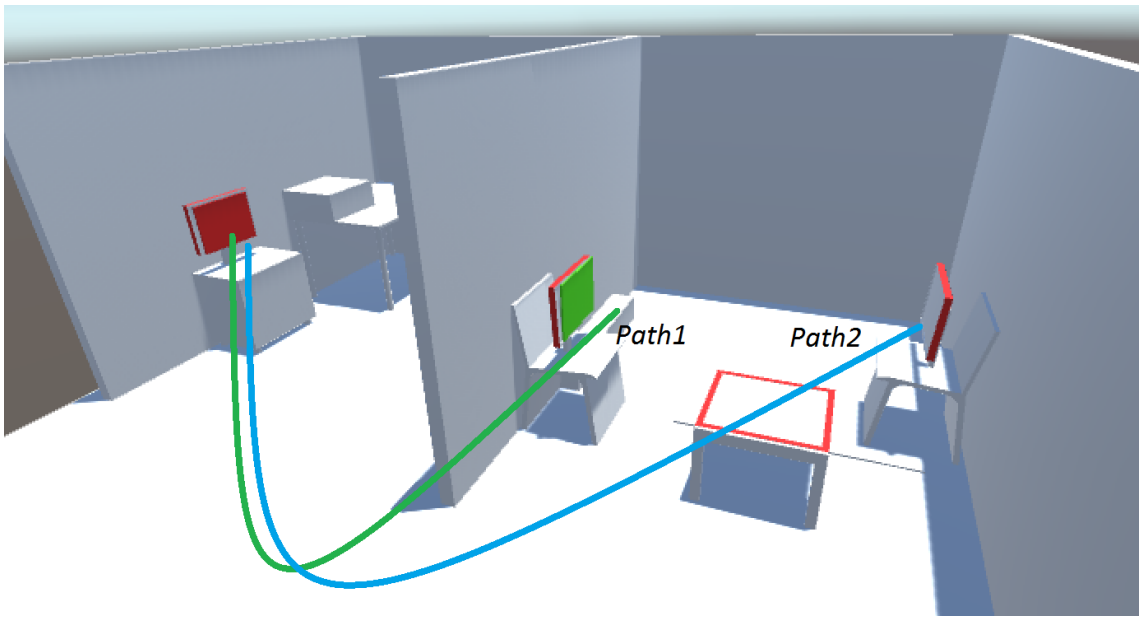


Figure 5.10: By altering the position of the green monitor, two different flight paths are generated per user.

arbitrary distance. During the training, users were informed with the positions of the red and green monitors in the 3D environment model. The environment model contains three physical monitors, but only two of them were active at any time in order to necessitate different flight paths (Figure 5.10). The time spent to read from each monitor was recorded as soon as the user reported the text correctly. We asked participants to use the gaze-to-see interaction technique, and we suggested to additionally use the overview-and-detail technique in EXO.

Positioning of the drone In this task, participants were expected to position the drone at three known target locations, which were visualized as boxes in the 3D models shown in both interfaces (Figure 5.9b and Figure 5.11). We logged the time spent to visit the target locations, whenever the system reported that the drone approached a target to within 10 cm tolerance. As the task involved accurate and fast positioning of the drone for this tasks, we suggested to the EXO users to use the pick-and-place technique.

Participants Ten participants (0 female, $\overline{X}$ =23.1 (sd=2.07) years old) volunteered in our experiment. All of them had extensive experiences with mobile devices, none was a regular drone pilot.

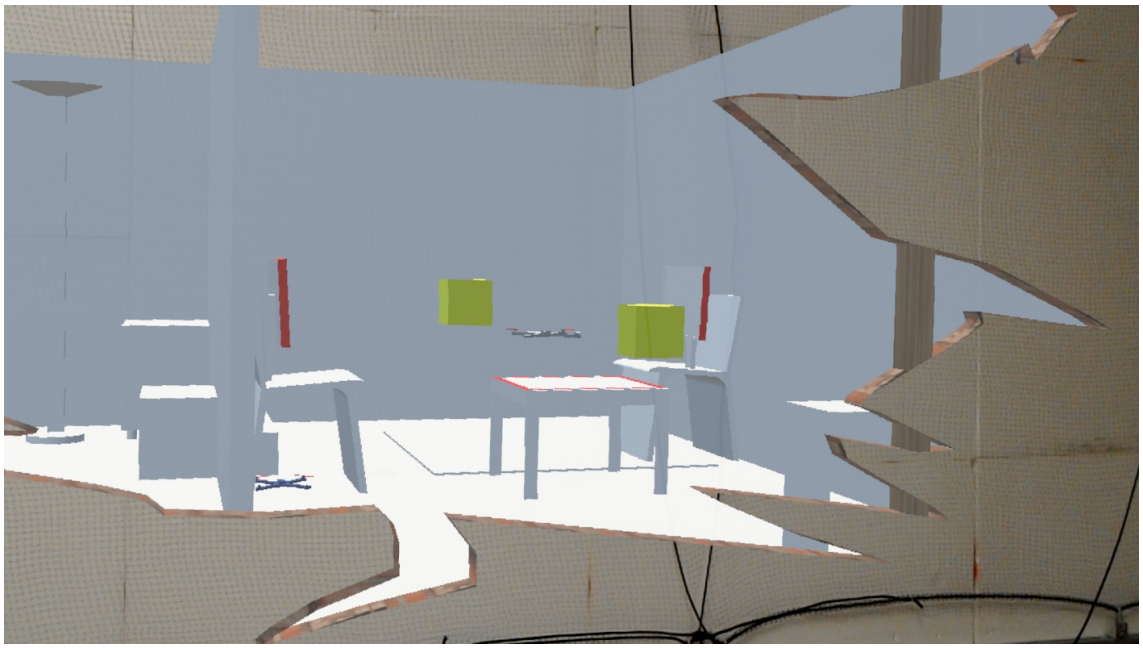


Figure 5.11: As part of positioning the drone task, target positions are visualized as yellow boxes in the EXO interface.

Experimental setup Participants performed the tasks while standing in front of a wall completely occluding the flight zone. In the EXO condition, participants wore a HoloLens for seeing through the wall. In the EGO condition, a joypad was used to steer the drone, while a monitor (19 inch) was used to display the video stream delivered by the camera of the drone. EGO users were also provided with 3D views of the flight zone from different perspectives (top and top-side view), displayed on a second monitor (15 inch) (Figure 5.9). A laptop was used to record the participants' qualitative and quantitative input during the experiment.

Procedure Participants were brought to the participant zone and informed about the setup of the experiment environment without giving detailed information about the flight zone. After the briefing, we assessed their demographics and explained how to use both interfaces. Participants were allowed to practice both interfaces, until they expressed confidence to use them.

Participants were asked to accomplish the tasks in randomized order, to eliminate training effects. For the text reading task, the position of the green monitor was changed to alter the flight path from the first to the second condition. After finishing each task with one interface, participants filled in the NASA *TLX*. Upon the completion of all tasks for both interfaces, participants filled out a

preference questionnaire, and a semi-structured interview was conducted. Sessions lasted 50 min.

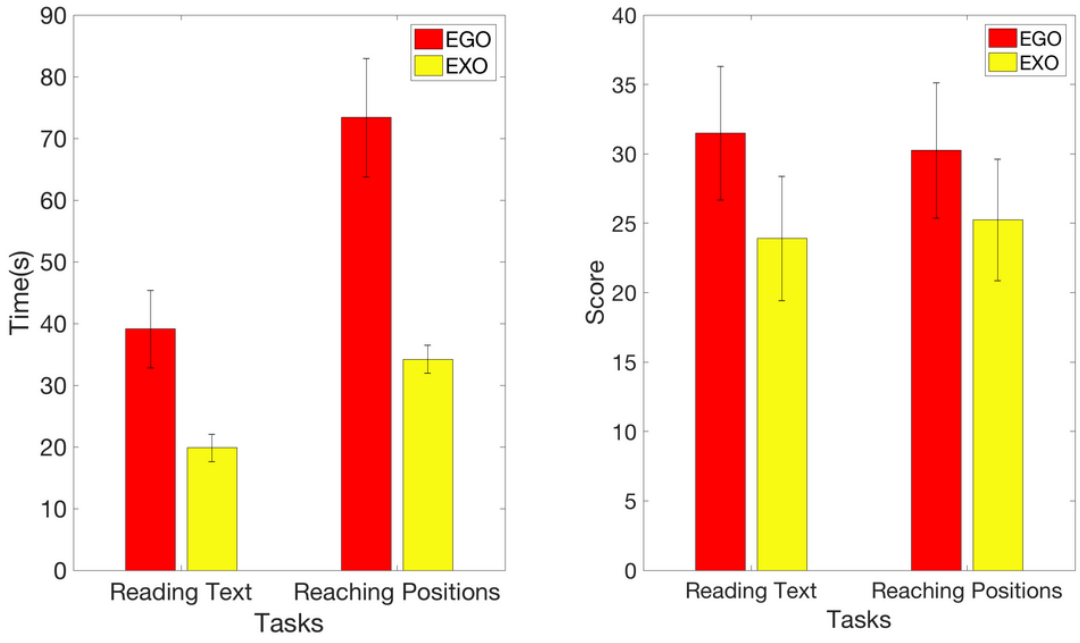
Results The task completion time was evaluated using paired t-tests, and the *TLX* data was analyzed using pairwise Wilcoxon signed-rank tests. The t-tests revealed significant differences between HoloLens and Joystick interface for both, the reading task (p=0.0013) and the reaching positions task (p=0.0024), in terms of task completion time. On average, task completion time of EXO was less than half of the average time required by the EGO (Figure 5.12a). In the text reading task, EXO took 19.85 seconds on average (standard error 2.2 seconds) to read the texts on both monitors, whereas EGO took 39.1 seconds on average (standard error 6.3 seconds). For reaching the given 3D positions, EXO users completed the task on average in 34.2 seconds (standard error 2.3 seconds). EGO took 73.4 seconds on average (standard error 9.57 seconds).

According to the overall scores of the NASA *TLX* forms, for both of the tasks, users found EXO to have a slightly better usability than EGO. For the first task, users gave an average score of 24 for EXO and 32 for EGO, whereas, for the second task, EXO scored 25 and EGO scored 30 (Figure 5.12b). Probably due to the small number of participants, the *TLX* data did not show significant differences between the interfaces. However, we found a noticeable trend in the *TLX* data towards the HoloLens interface for the reading task (Z=1.68, p=0.105).

Relatively high deviations in task completion time of EGO suggest that EGO requires a good 3D interpretation or experience with joypad control. In contrast, EXO seems to efficiently leverage human abilities, resulting in consistent performance, specifically for pick-and-place.

In the informal feedback during the post-interview, users commented on their preferences. All the participants stated that they would prefer EXO for the given tasks or similar task for investigation of the occluded space. Verbal feedback from the interviews for both conditions included:

- I felt more confident of being precise when using EXO, specifically using pick-and-place.
- I was feeling inside the scene with EXO.
- Depth feeling was amazing with EXO.
- I confused my orientation with EGO.
- I couldn't decide which view to concentrate on with EGO.
- Pick-and-place was cool, natural and accurate.
- Observing the drone from a distance, but still being able to get close to it, was pleasant.



Time Measurements NASA *TLX* scores

Figure 5.12: Task completion times and scores using EGO and EXO interfaces. a) Average time spent on the tasks with our two interfaces. EXO users performed much faster in both of the tasks, with similar performance, as indicated by the standard error. b) NASA *TLX* scores of the both interfaces for the given tasks. Results indicate that the time to complete a given task and the demanded cognitive load using EXO interface is less than EGO interface.

On EGO, several users commented that the joypad axis confusion between drone's local frame and global frame during steering was difficult. They also sometimes confused buttons, a problem that may be overcome with longer training. Nonetheless, the direct manipulation in EXO was more easily adopted. Users also criticized the limited field of view of EGO and reported a confusion of heights. Finally, they found that they could not easily decide which view (camera image or perspective views) to concentrate on.

On EXO, one user stated he preferred the precision of the joypad interface for collecting boxes, and several users found the HoloLens pick gesture inconvenient. However, both comments were likely caused by the unreliable gesture detection provided on the HoloLens. We hope that a future update of the HoloLens SDK will include a more stable gesture detection, which directly will make our pick-and-place interface appear more convenient and more precise. In summary, the results of our experiment allow to accept H_1 and H_2. Furthermore, we partially accept H_3 based on the trend towards EXO provided by the user comments and the data retrieved from *TLX* questionnaires.

5.3.2 Virtual Viewpoint Study

The physical viewpoint experiment demonstrated the use of exocentric interaction techniques at close distances. If the drone is further away from the user, drone control by hand gestures obviously becomes increasingly sensitive to fine-motor control of the hand and to tracking errors. We empirically verified that, indeed, satisfactory drone control with gestures is not possible at distances of 20 m or more.

However, since our exocentric (X-ray) interface uses the physical environment – the brick wall – only to provide relative motion cues to the user, a Virtual Reality (VR) interface using the same setup is also possible. In *VR*, the *HMD* is operated in a non-see-through mode, and the user is placed in a purely virtual environment, with the exception of the texture-mapped remote video stream. This setup can always place the user's virtual viewpoint in convenient proximity to the drone to allows direct manipulation. The *VR* interface is also necessary if physical proximity to the drone is not possible, for example, in dangerous environments.

We speculated that the virtual viewpoint interface would perform similar to the physical viewpoint interface (the latter is essentially the same as EXO in the previous experiment). We formulated our hypotheses as follows:

H_4: *"Users will perform similar in terms of execution time for a virtual viewpoint as for a physical viewpoint"*

H_5: *"A virtual viewpoint does not affect how a user completes the tasks, while being away from the scene"*

We tested these hypotheses by repeating the previous experiment with virtual viewpoint and physical viewpoint conditions, as follows.

Procedure In virtual viewpoint, participants performed the tasks while standing completely away from the occluded space. The visor of the HoloLens was entirely covered with a blinder to disable its see through display nature and turn it into a *VR* device. At the beginning of the experiment, virtual viewpoint users witnessed an animated camera transition from their current physical viewpoint to the virtual viewpoint at the remote location. The animation gave them the impression of flying to the target zone and landing where they had to perform the experiment.

In contrast, physical viewpoint users were standing just behind the occluding brick wall like in the physical viewpoint study. Compared to the first study, we had a slightly larger flight space with the same floor plan characteristics. In the virtual viewpoint study, again ten participants (0 female,

$\overline{X}$ =27.5 (sd=2.33) years old) volunteered in our experiment. All of them had extensive experiences with mobile devices, none was a regular drone pilot (different subjects from the physical viewpoint study).

Results In the text reading task, the physical viewpoint condition took 48.62 seconds on average (standard error 2.5 seconds) to read the texts on both monitors, whereas the virtual viewpoint condition took 43.37 seconds on average (standard error 2.9 seconds). For reaching the given 3D positions, physical viewpoint users completed the task on average in 44.03 seconds (standard error 3.72 seconds). virtual viewpoint took 41.21 seconds on average (standard error 1.53 seconds). It should be noted that flight times are slightly increased compared to the first study due to the enlarged space and longer paths.

According to the overall scores of the NASA *TLX* forms, for both of the tasks, users found physical viewpoint to have a slightly better usability than virtual viewpoint. For the first task, users gave an average score of 23 for physical viewpoint and 26 for virtual viewpoint, whereas, for the second task, physical viewpoint scored 25 and virtual viewpoint scored 27. While users commented to perceive both systems as almost identical for completing the tasks, they reported to prefer the physical viewpoint condition more due to its see-through visualization capability.

The results let us accept H_4 and H_5.

5.4 Discussion

We propose using real-scale interactions for steering remote drones. This enables simple control of the drone with low cognitive effort. Based on the feedback of users and the quantitative results of our experiments, we believe that pick-and-place interaction is useful for quickly positioning the drone when fully automatic navigation is not enough. While wearing the *HMD*, users have stereo vision to perceive depth. In addition, users can quickly change their viewpoint by simply moving around in a natural way to understand where an object is located in 3D. In contrast, a traditional desktop interface requires several scene manipulations to understand the 3D position of an object in the scene, especially when the object is floating in the air. Simple and natural exploration of the position of the drone in 3D space enables quick understanding of spatial relations, which is a fundamental requirement for navigating the drone in 3D.

Our pick-and-place technique uses a single target point to position the drone. While we could continuously sample points along a path defined by the user, we restrict the number of waypoints to a single start point and end point to ensure a precise placement and to avoid unnecessary drone

motion. Mapping any user motion directly to the position of the drone would not allow the user to search for the final position, while the drone follows the user's hand motion.

While pick-and-place can be used to precisely place a drone in 3D space, the gaze-to-see technique can be used to continuously explore and search the environment. Gaze-to-see is a high-level, goal-oriented interaction between drone and human with low cognitive requirements. It provides a tool for quickly observing a region of interest without dealing with positioning the drone.

Both of our interaction techniques outperform the traditional egocentric interface for controlling a drone. Note that the significant time difference observed between our experimental conditions are not the result of different reaction times, such as the time spent on moving the head when wearing an *HMD* versus pressing a button on joypad. The differences can rather be largely attributed to the user's efforts towards fine-tuning the position of the drone to solve the task. For example, finding the correct pose for the drone to read a small text, while experiencing motion blur during the movement phase, takes more time with EGO. In contrast, EXO users can easily assume a convenient pose thanks to the gaze-to-see technique.

Apart from the motion blur, no text rendering artifacts were disturbing the EGO users, as can be seen in Figure 5.9a. In contrast, the EXO users experienced both motion blur and slight artifacts due to the limited resolution of the *HMD* (Figure 5.6). We expect that with better *HMD*, quality the advantages of EXO may even become more pronounced.

Similarly, during the positioning drone task, EGO users had difficulties understanding if the drone was at the correct position from the given perspective views, whereas EXO users quickly identified the right position by virtue of the stereoscopic view.

Despite the good performance of EXO, we noticed a number of limitations during the experiments, which we describe in the following, along with recommendations for overcoming them based on our experience with the system.

Limited resolution. Our placement precision depends on the distance. As the drone moves away from the user, the increased distance affects the precision of pick-and-place. In addition, when the surface is far away from the user, it is hard to gaze at it. This provides a challenge for selecting the drone with pick-and-place interaction, and it makes it harder to position the drone in front of the right surface during gaze-to-see interaction. This limitation arises from the fact that humans cannot keep their head stable at millimeter-level accuracy. These limitations are solved when the user is virtually teleported to a viewpoint close to the drone, as demonstrated by our virtual viewpoint study. In fact, the virtual viewpoint technique can be seen as a generalization of the

overview-and-detail technique. The user can always use the virtual viewpoint mode to virtually move closer to the drone and thus increase the precision. The blinder on the visor may not even be necessary, as implied by users' preferences.

Projection error. When the outside in tracking is not precise enough, misregistration causes the projected images to not line up properly with the 3D model. In addition, if the poses are not synchronized with the camera images, the error is further increased. However, these problems can be overcome by better tracking, ideally incorporating dense reconstructions obtained in real time from a drone equipped with suitable sensors, such as structure-from-light sensors or stereo cameras.

Tracking error. Depending on the tracking accuracy of the system, the drone may position itself slightly off the target destination, although the results would still be visualized as if the drone was at the correct location. During tasks requiring accurate spatial positioning, such as drilling a hole at the right spot, the user may be misled. A hybrid interface showing both the exocentric synthetic view and the egocentric video stream side by side may partially alleviate this problem.

Reconstruction error. Gaze-to-see can be strongly affected by a wrongly estimated surface normals, if the 3D model is automatically reconstructed using structure from motion algorithms. However, many exploration tasks do not require photorealistic rendering and tolerate heavy low-pass filtering of normals to suppress unwanted outliers.

Eye calibration error. Like any ray-picking technique, pick-and-place performance is affected by eye calibration. Without a good estimation of the eye position, any deviations of physical eye and virtual camera will be magnified by the projected distance, letting the picked virtual position drift from the hand after some displacement. During our experiments, we noticed that users coped with such situations by simply releasing their grip and quickly re-picking the drone, essentially improvising a form of clutching to minimize the aggregation of unwanted drift.

3D interaction. The mathematics of scaled-world-grab [92] imply that when the user moves an object away from the body, movement precision will drop quickly. As a remedy, users can re-adjust their virtual viewpoint to move closer to the target location or look at the drone from a different perspective to control the drone more precisely. Likewise, if surfaces face away from the user, gaze-to-see requires first assuming a rotated virtual viewpoint to look at the target position.

Aerodynamic restrictions. A drone's aerodynamics restrict it from quickly adapting into a new given position. Therefore, gaze-to-see interaction technique had to be limited to a fixed number of position commands instead of continuous ones where a new position command is sent each time the user looks to a different surface point.

Selecting the remote scene. A virtual viewpoint is natural for immersive *VR* users, while *AR* user must switch from their physical viewpoint to a virtual one. This can lead to confusion between real and virtual objects. The overview-and-detail technique mostly avoids such confusion, but introduces the restriction that users can only move closer to a point they are already gazing at. While this is sufficient for a number of tasks, choosing a new viewpoint relative to gaze has clear limitations. In particular, gazing becomes less precise and more difficult at larger distances.

However, common techniques such as world-in-miniature [125] (WIM) can be used to overcome this limitation. Using a gesture, users can obtain a miniaturized copy of the scene in front of them, in the same orientation as their current viewpoint. It is straightforward to apply scene manipulation techniques from traditional desktop interfaces to a WIM. Users can rotate the WIM towards the desired view and apply clipping or transparency to expose interior structures. They can apply exocentric selection of movement targets in the WIM rather than in the egocentric perspective. In case of a rescue operation, the use of a WIM naturally extends towards a remote control center overview of multiple drones and rescuers from an exocentric perspective.

5.5 Summary

We have developed a prototypical system to discover a remote or occluded scene in an intuitive way by visualizing live imagery streamed from a camera drone in a three-dimensional, exocentric context. To control the exploration, we have implemented experimental high-level interaction techniques that control the drone indirectly, by relating to the enclosing space in which the drone is flying rather than the drone's own local coordinate system and flight parameters, such as speed or altitude. This gives the user the impression of being present next to the drone, or having X-ray vision when using a see-through display. Our experiments confirm that this style of interaction is efficient compared to conventional remote piloting and that it is attractive for users.

6 Conclusions

Contents

6.1 Summary

In this thesis, we presented a set of Image-Based Modeling and Rendering (IBMR) techniques and novel user interactions powered by *IBMR* for remote exploration problem. Our techniques contribute to the field of tele-exploration from a user-centered perspective. Specifically, we aimed to reduce the cognitive load of users by presenting them a photorealistic rendering. In addition, depending on the remote exploration scenario, the users may be given interaction capability with the remote environment. Hence, we also combined *IBMR* with a set of intuitive interaction techniques to minimize the cognitive load. We further contribute to remote exploration field by enhancing *IBMR* with our spatio-temporally coherent inpainting method.

When combined with an existing Structure from Motion (SfM) algorithm, our techniques can run as a whole pipeline which addresses all the needs of a tele-exploration task: remote data acquisition and interaction, SfM, texturing and inpainting of the incomplete geometry (see Figure 6.1).

Remote data acquisition and interaction In a tele-exploration task data has to be acquired remotely, preferably using highly mobile and easy to control robots. Sensory data collected from such robots can be used by SfM algorithms and forwarded into our *IBMR* algorithm as proxy geometry and images with pose information for texturing. To achieve this, our Drone-augmented

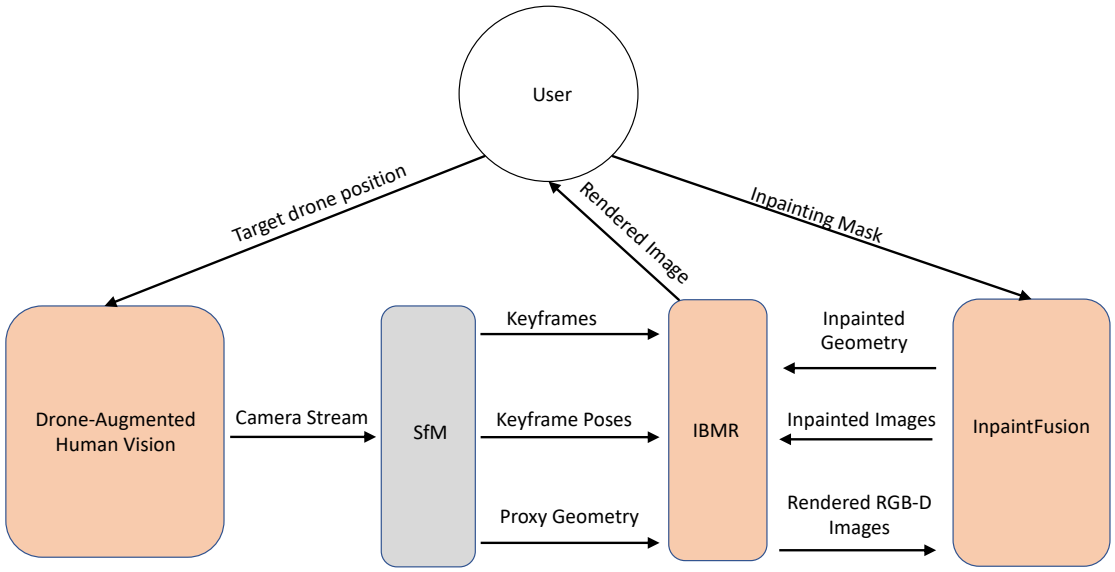


Figure 6.1: By adding an existing structure from motion (SfM) algorithm (gray colored), our components (orange colored) can be combined to create a whole tele-exploration application pipeline. Order of events: (1) User sets a target position using our interactions for the drone, (2) drone streams camera images from the target position, (3) an existing SfM algorithm generates needed input for our *IBMR* system, (4) user receives rendered image by *IBMR*, (5) User marks pixels for inpainting and creates a mask, (6) InpaintFusion inpaints the depth and RGB images within the masked region, (7) *IBMR* imports the new geometry and inpainted images, (8) user observes updated rendering.

human vision work can be utilized to provide data for SfM algorithms. Despite Drone-augmented human vision also providing an image based rendering, the rendering is limited only to the latest image acquired from onboard camera. Therefore, combining it with our *IBMR* system would ensure texturing all parts of the discovered geometry. In addition, considering the remote exploration problem as a whole, we introduce novel interactions powered by *IBMR*. *IBMR* gives users the flexibility to change their viewpoint independent of the sensor at the remote location. We utilized this to the full extent and let users control a drone from an exocentric point of view using a Head-Mounted Display (HMD). As discussed in Chapter 5, an exocentric point of view helps users position a drone more accurately and in a timely manner which is essential for data acquisition.

Texturing Despite abundance of various *IBMR* algorithms, none of the existing methods targets a remote exploration scenario. Their lack of memory management and computational expense

makes them rather suitable for offline applications. Telepresence applications, especially the tele-exploration task, require real-time feedback from the remote location, which only can be achieved with an online system that can be coupled with a SfM algorithm. In addition, as the name asserts, exploration applications require no prior knowledge of the scene being investigated. Therefore, spatial distribution of available memory to the scene in advance is not an option. Our system handles this problem by a novel online view planning algorithm. On the fly, it selects the required images needed for texturing. Moreover, it optimizes their color intensity and poses before projecting them into the scene, such that the resulting rendering has minimal to no color tone difference or blurring.

Inpainting Due to the lack of a perfect reconstruction system, 3D scans can have holes of various sizes in their resulting 3D model. For a good user experience, having holes in a remote exploration scenario is not desirable. Our spatio-temporally coherent inpainting method can process the rendering results of our *IBMR* pipeline and fill in the holes in 3D space (see Figure 4.16 and 4.17). Resulting inpainted geometry can be forwarded into our *IBMR* pipeline. Existing inpainting techniques either inpaint in 2D image space or introduce a dominant plane to cover the holes, which is not suitable for handling occlusions or for integrating into an *IBMR* pipeline.

6.1.1 Contributions

Previous to the writing of this thesis, *IBMR* techniques were never applied for a tele-exploration task. Therefore, online photorealistic rendering of the discovered environment was not possible. Operators of the tele-exploration robots had to switch to first person camera view for seeing details and to per vertex colored 3D reconstruction view to have a 3D understanding of the scene. We combined these two views into one by photorealistic rendering and reduced the cognitive demand of switching and relating between two different views. In addition, operators were limited to control the robots using interaction techniques such as joystics that have high cognitive demands. Training such skilled operators is time intensive, costly and yet still has a higher risk of failing during and exploration mission due to high cognitive demand. Furthermore, we let users inpaint parts of the scene while discovering the remote environment. Online editing of the scene would be necessary to decide if further images has to be captured from the scene while still on the mission.

6.1.2 Future Work and Limitations

As future work, missing SfM component in our pipeline can be implemented from an existing SfM algorithm. To combine Drone-augmented human vision system with our desktop powered *IBMR*

system, *HMD* can directly use desktop computation power and we can eliminate the use of a 2D display in our *IBMR* system. In the current implementation, target region mask to inpaint on an image can only be labeled in 2D image space. This introduces a limitation to *HMD* usage. To allow labeling from a *HMD*, users could define voxel regions to be inpainted using their hand gestures like pointing as in pick-and-place interaction. *IBMR* system would be configured to receive mesh and keyframe updates from InpaintFusion. Finally, limitations of each of our individual components discussed in chapters could be addressed.

6.2 Outlook

With the rapid progress of computer vision research and availability of new hardware, in the near future, 3D reconstructions will require only a few modifications by the inpainting techniques to fill in the missing geometry. Using improved artificial intelligence techniques, inpainting will complete the scene in a semantic and more realistic manner. Camera pose errors will be decreased to a lesser extent. Therefore, we will see blur-free blending of the images on the geometry. Thanks to faster video cards, it will be possible to simultaneously process more images from multiple input sources. Superior features of offline *IBMR* techniques could be also implemented in our online method. With the advancements in *HMD* technology, it will be possible to blend multiple textures for the rendering purposes.

Summarizing the findings of this thesis, we conclude that *IBMR* techniques contribute to the remote exploration field by offering photorealistic renderings. It still remains as an open and interesting research field that can contribute to important applications, like search and rescue scenarios, surveillance and space exploration.

A

List of Acronyms

Glossary

6DoF Six Degree of Freedom

AR Augmented Reality

DR Diminished Reality

FOV Field of View

GAN Generative Adversarial Network

GPU Graphics Processing Unit

HMD Head-Mounted Display

IBMR Image-Based Modeling and Rendering

IMU Inertial Measurement Unit

MAV Micro Aerial Vehicle

MR Mixed Reality

NTP Network Time Protocol

ROI Region of Interest

ROS Robot Operating System

SLAM Simultaneous Localization and Mapping

TLX Task Load Index

TSDF Truncated Signed Distance Function

UDP User Datagram Protocol

VR Virtual Reality

B Videos

B.1 Related to Chapter 3

Related publication: O. Erat, M. Hoell, K. Haubenwallner, C. Pirchheim and D. Schmalstieg, "Real-Time View Planning for Unstructured Lumigraph Modeling," *in IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 11, pp. 3063-3072, Nov. 2019

Figure B.1: Available at: https://youtu.be/LCL4pv7Klpw

B.2 Related to Chapter 4

Related publication: Mori,Shohei; Erat, Okan; Broll, Wolfgang; Saito, Hideo; Schmalstieg, Dieter; Kalkofen, Denis InpaintFusion: Incremental RGB-D Inpainting for 3D Scenes

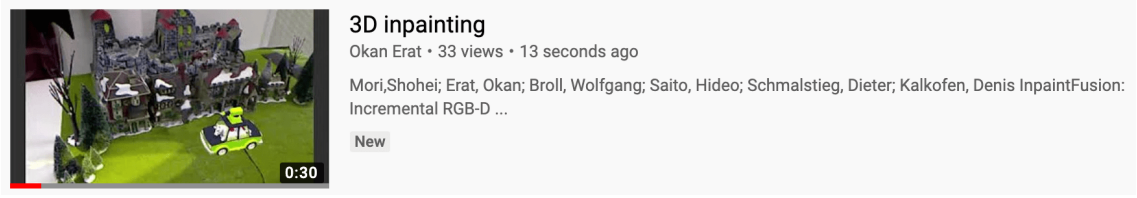


Figure B.2: Available at: https://youtu.be/byl-QVreGYg

B.3 Related to Chapter 5

Related publication: O. Erat, W. A. Isop, D. Kalkofen and D. Schmalstieg, "Drone-Augmented Human Vision: Exocentric Control for Drones Exploring Hidden Areas," *in IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 4, pp. 1437-1446, April 2018

Figure B.3: Available at: https://youtu.be/drwgdPe7VzE

Bibliography

[1] Avery, B., Piekarski, W., and Thomas, B. H. (2007). Visualizing occluded physical objects in unfamiliar outdoor augmented reality environments. In *Proceedings of ISMAR*, pages 1–2, Washington, DC, USA. (page 23)

[2] Avery, B., Sandor, C., and Thomas, B. H. (2009). Improving spatial perception for augmented reality x-ray vision. In *Proc. IEEE VR*, pages 79–82. (page 23)

[3] Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. (2009). PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 28(3). (page 18, 59)

[4] Barnes, C., Shechtman, E., Goldman, D. B., and Finkelstein, A. (2010). The Generalized PatchMatch Correspondence Algorithm. In *European Conf. on Computer Vision (ECCV)*, pages 29–43. (page 5, 18)

[5] Bergé, L.-P., Aouf, N., Duval, T., and Coppin, G. (2016). Generation and VR visualization of 3D point clouds for drone target validation assisted by an operator. In *Computer Science and Electronic Engineering (CEEC), 2016 8th*, pages 66–70. IEEE. (page 22)

[6] Bi, S., Kalantari, N. K., and Ramamoorthi, R. (2017). Patch-based optimization for image-based texture mapping. *ACM Transactions on Graphics*, 36(4):1–11. (page 5, 7, 14)

[7] Billinghurst, M., Kato, H., and Poupyrev, I. (2001). The MagicBook - Moving Seamlessly between Reality and Virtuality. *IEEE Computer Graphics and Applications*, 21(1):6–9. (page 81, 84)

[8] BostonDynamics, L. (2017). Legged squad support systems. (page 13)

[9] Buehler, C., Bosse, M., McMillan, L., Gortler, S., and Cohen, M. (2001a). Unstructured lumigraph rendering. In *Proceedings SIGGRAPH*, pages 425–432. (page 8, 15, 28, 29)

[10] Buehler, C., Bosse, M., McMillan, L., Gortler, S., and Cohen, M. (2001b). Unstructured lumigraph rendering. In *Proc. Annual Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 425–432. (page 62)

[11] Cayon, R. O., Djelouah, A., and Drettakis, G. (2015). A Bayesian approach for selective image-based rendering using superpixels. In *3DV*, pages 469–477. (page 15)

[12] Chai, J.-X., Chan, S.-C., Shum, H.-Y., and Tong, X. (2000). Plenoptic sampling. In *Proc. SIGGRAPH*, pages 307–318. ACM Press. (page 14)

[13] Chaurasia, G., Duchêne, S., Sorkine-Hornung, O., and Drettakis, G. (2013). Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics*, 32. (page 15)

[14] Chen, J. Y., Haas, E. C., and Barnes, M. J. (2007). Human performance issues and user interface design for teleoperated robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 37(6):1231–1245. (page 20)

[15] Chen, W.-C., Bouguet, J.-Y., Chu, M. H., and Grzeszczuk, R. (2002). Light field mapping: Efficient representation and hardware rendering of surface light fields. In *Proc. SIGGRAPH*, pages 447–456. (page 15)

[16] Cho, K., Cho, M., and Jeon, J. (2016). Fly a drone safely: Evaluation of an embodied egocentric drone controller interface. *Interacting with Computers*. (page 21)

[17] Coffin, C. and Hollerer, T. (2006). Interactive perspective cut-away views for general 3D scenes. In *Proc. 3DUI*, pages 25–28. (page 23)

[18] Cosco, F., Garre, C., Bruno, F., Muzzupappa, M., and Otaduy, M. A. (2013). Visuo-haptic mixed reality with unobstructed tool-hand integration. *IEEE Trans. on Visualization and Computer Graphics*, 19(1):159–172. (page 17)

[19] Criminisi, A., Perez, P., and Toyama, K. (2004). Region filling and object removal by exemplar-based image inpainting. *IEEE Trans. on Image Processing*, 13(9):1200–1212. (page 18)

[20] Davies, R. (2001). Technology versus terrorism. *Jane's International Defence Review*, pages 36–43. (page 13)

[21] Davis, A., Levoy, M., and Durand, F. (2012a). Unstructured light fields. *Comput. Graph. Forum*, 31(2pt1):305–314. (page 8, 15)

[22] Davis, A., Levoy, M., and Durand, F. (2012b). Unstructured light fields. *Computer Graphics Forum*, 31(2):305–314. (page 62)

[23] Debevec, P. (1998). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proc. Annual Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 189–198. (page 18)

[24] Debevec, P., Taylor, C., and Malik, J. (1996). Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proc. SIGGRAPH*, pages 11–20. (page 15)

[25] Dhamo, H., Tateno, K., Laina, I., Navab, N., and Tombari, F. (2019). Peeking behind objects: Layered depth prediction from a single image. *Pattern Recognition Letters*, 125:333 – 340. (page 19)

[26] Dong, S. and Hollerer, T. (2018). Real-Time Re-Textured Geometry Modeling Using Microsoft HoloLens. In *IEEE Virtual Reality*, pages 231–237. (page 14)

[27] Dou, M. and Fuchs, H. (2014). Temporally enhanced 3D capture of room-sized dynamic scenes with commodity depth cameras. In *2014 IEEE Virtual Reality (VR)*, pages 39–44. IEEE. (page 15)

[28] Eisemann, M., De Decker, B., Magnor, M., Bekaert, P., de Aguiar, E., Ahmed, N., Theobalt, C., and Sellent, A. (2008). Floating Textures. *Computer Graphics Forum*, 27(2):409–418. (page 15)

[29] Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In *European conference on computer vision*, pages 834–849. Springer. (page 13)

[30] Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg. Springer-Verlag. (page 36)

[31] Fehn, C. (2004). Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In *Stereoscopic Displays and Virtual Reality Systems XI.*, volume 5291, page 1. Proc. Int. Society for Optics and Photonics. (page 18)

[32] Feiner, S. K., Webster, A. C., Krueger, T. E., MacIntyre, B., and Keller, E. J. (1995). Architectural anatomy. *Presence*, 4(3):318–325. (page 23)

[33] Fleishman, S., Cohen-Or, D., and Lischinski, D. (1999). Automatic camera placement for image-based modeling. In *Proc. Pacific Conference on Computer Graphics and Applications*, pages 12–20. (page 8, 15)

[34] Forster, C., Lynen, S., Kneip, L., and Scaramuzza, D. (2013). Collaborative monocular SLAM with multiple micro aerial vehicles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3962–3970. IEEE. (page 14)

[35] Fuhrmann, S., Langguth, F., and Goesele, M. (2014). Mve-a multi-view reconstruction environment. In *GCH*, pages 11–18. (page 37)

[36] Gal, R., Wexler, Y., Ofek, E., Hoppe, H., and Cohen-Or, D. (2010). Seamless montage for texturing models. *Eurographics 2010*, 29(2). (page 7, 14)

[37] Gauglitz, S., Nuernberger, B., Turk, M., and Höllerer, T. (2014). World-stabilized annotations and virtual scene navigation for remote collaboration. In *ACM UIST*, pages 449–459. (page 15)

[38] Godard, C., Mac Aodha, O., and Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. In *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. (page 19)

[39] Goertz, R. C. (1953). Remote-control manipulator. US Patent 2,632,574. (page 3)

[40] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. In *Proc. Advances in Neural Information Processing Systems*, pages 2672–2680. (page 19)

[41] Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The lumigraph. In *Proc. SIGGRAPH*, pages 43–54. (page 7, 14)

[42] Gruber, L., Richter-Trummer, T., and Schmalstieg, D. (2012). Real-time Photometric Registration from Arbitrary Geometry. In *ISMAR*. (page 34)

[43] Guo, K., Xu, F., Yu, T., Liu, X., Dai, Q., Liu, Y., Guo, K., Xu, F., Yu, T., Liu, X., Dai, Q., and Liu, Y. (2017). Real-Time Geometry, Albedo, and Motion Reconstruction Using a Single RGB-D Camera. *ACM Trans. Graph.*, 36(3):1–13. (page 15)

[44] Hansen, J. P., Alapetite, A., MacKenzie, I. S., and Møllenbach, E. (2014). The use of gaze to control drones. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 27–34. ACM. (page 20)

[45] Hart, S. G. and Staveland, L. E. (1988). Development of NASA-TLX (task load index): Results of empirical and theoretical research. *Human mental workload*, 1(3):139–183. (page 93)

[46] Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press. (page 17)

[47] Hashimoto, S., Ishida, A., Inami, M., and Igarashi, T. (2011). Touchme: An augmented reality based remote robot manipulation. In *Proc. ICAT*. (page 22)

[48] Hedman, P., Ritschel, T., Drettakis, G., and Brostow, G. (2016). Scalable inside-out image-based rendering. *ACM Trans. Graph.*, 35(6):1–11. (page 5, 8, 15)

[49] Herling, J. and Broll, W. (2010). Advanced self-contained object removal for realizing real-time diminished reality in unconstrained environments. In *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, pages 207–212. (page 16, 18, 64)

[50] Herling, J. and Broll, W. (2014). High-quality real-time video inpainting with PixMix. *IEEE Trans. on Visualization and Computer Graphics*, 20(6):866–879. (page 16, 17, 18, 50, 51, 54, 55, 60, 64, 65, 73)

[51] Hewish, M. (2001). Gi, robot robots are taking over dirty, dangerous and dull missions. *Janes International Defense Review*, 34(1):34–40. (page 13)

[52] Higuchi, K. and Rekimoto, J. (2013). Flying head: a head motion synchronization mechanism for unmanned aerial vehicle control. In *CHI'13 Extended Abstracts*, pages 2029–2038. ACM. (page 20)

[53] Hing, J. T., Sevcik, K. W., and Oh, P. Y. (2010). Development and evaluation of a chase view for uav operations in cluttered environments. *Journal of Intelligent & Robotic Systems*, 57(1):485–503. (page 22)

[54] Hlaváč, V., Leonardis, A., and Werner, T. (1996). Automatic selection of reference views for image-based scene representations. pages 526–535. Springer, Berlin, Heidelberg. (page 15)

[55] Hoang, T. N. and Thomas, B. H. (2010). Augmented viewport: An action at a distance technique for outdoor ar using distant and zoom lens cameras. In *Proc. ISWC*, pages 1–4. (page 24)

[56] Holynski, A. and Kopf, J. (2018). Fast depth densification for occlusion-aware augmented reality. *ACM Trans. on Graphics*, 37(6):194:1–194:11. (page 18)

[57] Hoppe, C., Klopschitz, M., Rumpler, M., Wendel, A., Kluckner, S., Bischof, H., and Reitmayr, G. (2012). Online feedback for structure-from-motion image acquisition. In *BMVC*. (page 8, 15)

[58] Hornung, A., Zeng, B., and Kobbelt, L. (2008). Image selection for improved multi-view stereo. In *CVPR*. (page 8)

[59] Huang, J., Dai, A., Guibas, L., and Nießner, M. (2017). 3DLite: Towards Commodity 3D Scanning for Content Creation. *ACM Transactions on Graphics 2017 (TOG)*. (page 15)

[60] Iizuka, S., Simo-Serra, E., and Ishikawa, H. (2017). Globally and locally consistent image completion. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 36(4):107:1–107:14. (page 19)

[61] Ilan, S. and Shamir, A. (2015). A survey on data-driven video completion. In *Computer Graphics Forum*, volume 34, pages 60–85. Wiley Online Library. (page 17)

[62] Isogawa, M., Mikami, D., Takahashi, K., Iwai, D., Sato, K., and Kimata, H. (2018). Which is the Better Inpainted Image? Training Data Generation Without Any Manual Operations. *Int. Journal of Computer Vision*. (page 64)

[63] Isop, W. A., Pestana, J., Ermacora, G., Fraundorfer, F., and Schmalstieg, D. (2016). Micro aerial projector - stabilizing projected images of an airborne robotics projection platform. In *Proc. IROS*, pages 5618–5625. (page 85)

[64] Jachnik, J., Newcombe, R. A., and Davison, A. J. (2012). Real-time surface light-field capture for augmentation of planar specular surfaces. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 91–97. IEEE. (page 15)

[65] Kahler, O., Prisacariu, V. A., Ren, C. Y., Sun, X., Torr, P. H. S., and Murray, D. W. (2015). Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device. *IEEE TVCG*, 22(11). (page 31, 37)

[66] Kalkofen, D., Mendez, E., and Schmalstieg, D. (2009). Comprehensible Visualization for Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):193–204. (page 8)

[67] Kameda, Y., Takemasa, T., and Ohta, Y. (2004). Outdoor see-through vision utilizing surveillance cameras. In *Proc. ISMAR*, pages 151–160. (page 23)

[68] Karanam, C. R. and Mostofi, Y. (2017). 3D through-wall imaging with unmanned aerial vehicles using Wifi. In *Proc. IPSN*, pages 131–142. (page 22)

[69] Kasahara, S., Niiyama, R., Heun, V., and Ishii, H. (2013). exTouch: spatially-aware embodied manipulation of actuated objects mediated by augmented reality. In *Proc. ACM TEI*, pages 223–228. (page 21)

[70] Kasahara, S. and Rekimoto, J. (2014). JackIn: Integrating first-person view with out-of-body vision generation for human-human augmentation. In *Proc. Augmented Human*, pages 46:1–46:8. (page 23)

[71] Kawai, N., Sato, T., Nakashima, Y., and Yokoya, N. (2017). Augmented reality marker hiding with texture deformation. *IEEE Trans. on Visualization and Computer Graphics*, 23(10):2288–2300. (page 9, 16, 18, 54, 64)

[72] Kawai, N., Sato, T., and Yokoya, N. (2016). Diminished reality based on image inpainting considering background geometry. *IEEE Trans. on Visualization and Computer Graphics*, 22(3):1236–1247. (page 16, 18, 50, 51, 54, 55, 64, 65)

[73] Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., and Kolb, A. (2013). Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *Proc. Int. Conf. on 3D Vision*, pages 1–8. (page 52, 53, 54, 55)

[74] Klein, G. and Murray, D. (2007a). Parallel tracking and mapping for small AR workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10. IEEE Computer Society. (page 13)

[75] Klein, G. and Murray, D. (2007b). Parallel tracking and mapping for small AR workspaces. In *ISMAR*. (page 25)

[76] Klose, F., Wang, O., Bazin, J.-C., Magnor, M., and Sorkine-Hornung, A. (2015). Sampling based scene-space video processing. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 34(4). (page 17)

[77] Korkalo, O., Aittala, M., and Siltanen, S. (2010). Light-weight marker hiding for augmented reality. In *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, pages 247–248. (page 16, 54, 64)

[78] Kunert, C., Schwandt, T., and Broll, W. (2019). An efficient diminished reality approach using real-time surface reconstruction. In *Cyberworlds*. (page 18)

[79] Kusner, R., Kusner, W., Lagarias, J. C., and Shlosman, S. (2016). Configuration spaces of equal spheres touching a given sphere: The twelve spheres problem. *arXiv preprint arXiv:1611.10297*. (page 32)

[80] Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., and Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In *Proc. IEEE Int. Conf. on 3D Vision*, pages 239–248. (page 19)

[81] Lepetit, V., Berger, M., and Lorraine, L. (2001). An intuitive tool for outlining objects in video sequences: Applications to augmented and diminished reality. In *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, pages 159–160. (page 17)

[82] Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Proc. SIGGRAPH*, pages 31–42. (page 14)

[83] Li, C., Xiao, H., Tateno, K., Tombari, F., Navab, N., and Hager, G. D. (2016). Incremental scene understanding on dense slam. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. (page 54)

[84] Li, Z., Wang, Y., Guo, J., Cheong, L.-F., and Zhou, S. Z. (2013). Diminished reality using appearance and 3D geometry of internet photo collections. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 11–19. IEEE. (page 17, 54)

[85] Lichiardopol, S. (2007). A survey on teleoperation. *Technische Universitat Eindhoven, DCT report*, 20:40–60. (page 13)

[86] Low, K.-L. (2004). Linear least-squares optimization for point-to-plane icp surface registration. Technical Report TR04-004, Dept. of Computer Science, University of North Carolina, Chapel Hill. (page 54)

[87] Marty, P. et al. (2004). Alive: An autonomous light intervention vehicle. In *Advances In Technology For Underwater Vehicles Conference, Oceanology International*, volume 2004. (page 13)

[88] Meerits, S. and Saito, H. (2015). Real-time diminished reality for dynamic scenes. In *2015 IEEE International Symposium on Mixed and Augmented Reality Workshops*, pages 53–59. IEEE. (page 17)

[89] Meilland, M., Barat, C., and Comport, A. (2013). 3D High Dynamic Range dense visual SLAM and its application to real-time object re-lighting. In *ISMAR*, pages 143–152. (page 15, 42)

[90] Meka, A., Fox, G., Zollhofer, M., Richardt, C., and Theobalt, C. (2017). Live User-Guided Intrinsic Video for Static Scenes. *IEEE Transactions on Visualization and Computer Graphics*, 23(11):2447–2454. (page 15)

[91] Meka, A., Zollhöfer, M., Richardt, C., and Theobalt, C. (2016). Live intrinsic video. *ACM Transactions on Graphics*, 35(4):1–14. (page 15)

[92] Mine, M. R., Brooks Jr, F. P., and Sequin, C. H. (1997). Moving objects in space: exploiting proprioception in virtual-environment interaction. In *Proceedings SIGGRAPH*, pages 19–26. (page 82, 101)

[93] Mirk, D. and Hlavacs, H. (2014). *Using Drones for Virtual Tourism*, pages 144–147. Springer International Publishing, Cham. (page 20)

[94] Mori, S., Ikeda, S., and Saito, H. (2017). A survey of diminished reality: Techniques for visually concealing, eliminating, and seeing through real objects. *IPSJ Trans. on Computer Vision and Applications*, 9(17). (page 8)

[95] Morrison, J. G., Gálvez-López, D., and Sibley, G. (2016). Moarslam: Multiple operator augmented rslam. In *Distributed autonomous robotic systems*, pages 119–132. Springer. (page 14)

[96] Mulloni, A., Veas, E., Kruijff, E., and Schmalstieg, D. (2010). Techniques for view transition in multi-camera outdoor environments. In *Proc. Graphics Interface 2010*, Ottawa, Canada. (page 24)

[97] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). ORB-SLAM: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163. (page 13)

[98] Nakajima, Y., Mori, S., and Saito, H. (2017). Semantic object selection and detection for diminished reality based on SLAM with viewpoint class. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, pages 338–343. IEEE. (page 54, 74)

[99] Neumann, U., You, S., Hu, J., Jiang, B., and Lee, J. (2003). Augmented virtual environments (AVE): Dynamic fusion of imagery and 3D models. In *Proc. IEEE VR*, VR '03, pages 61–, Washington, DC, USA. (page 6, 10, 23)

[100] Newcombe, R. A., Fox, D., and Seitz, S. M. (2015). DynamicFusion : Reconstruction and Tracking of Non-rigid Scenes in Real-Time. In *CVPR*. (page 3)

[101] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). Kinectfusion: Real-time dense surface mapping and tracking. pages 127–136. (page 3, 14, 65)

[102] Nocerino, E., Poiesi, F., Locher, A., Tefera, Y. T., Remondino, F., Chippendale, P., and Van Gool, L. (2017). 3D reconstruction with a collaborative approach based on smartphones and a cloud-based server. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42(W8):187–194. (page 14)

[103] Orts-Escolano, S., Dou, M., Tankovich, V., Loop, C., Cai, Q., Chou, P., Mennicken, S., Valentin, J., Pradeep, V., Wang, S., Kang, S., Rhemann, C., Kohli, P., Lutchyn, Y., Keskin, C., Izadi, S., Fanello, S., Chang, W., Kowdle, A., Degtyarev, Y., Kim, D., Davidson, P., and Khamis,

S. (2016). Holoportation: Virtual 3D Teleportation in Real-time. In *ACM UIST*, pages 741–754. (page 3, 15, 29)

[104] Park, J. J., Newcombe, R., and Seitz, S. (2018). Surface light field fusion. In *Proceedings - 2018 International Conference on 3D Vision, 3DV 2018*, pages 12–21. IEEE. (page 15, 74)

[105] Pathak, D., Krähenbühl, P., Donahue, J., and Darrell, T. (2016). Context encoders: Feature learning by inpainting. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2536–2544. (page 19)

[106] Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 22(3):313–318. (page 61)

[107] Pfister, H., Zwicker, M., van Baar, J., and Gross, M. (2000). Surfels: Surface elements as rendering primitives. In *Proc. the Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 335–342, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. (page 53)

[108] Philip, J. and Drettakis, G. (2018). Plane-based multi-view inpainting for image-based rendering in large scenes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 1–11. (page 74)

[109] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*. (page 85)

[110] Rameau, F., Ha, H., Joo, K., Choi, J., Park, K., and Kweon, I. S. (2016). A real-time augmented reality system to see-through cars. *IEEE transactions on visualization and computer graphics*, 22(11):2395–2404. (page 54)

[111] Richter-Trummer, T., Park, J., Kalkofen, D., and Schmalstieg, D. (2016). Instant Mixed Reality Lighting from Casual Scanning. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'16)*, Merida, Mexico. (page 15, 36, 42)

[112] Rohmer, K., Büschel, W., Dachselt, R., and Grosch, T. (2015). Interactive near-field illumination for photorealistic augmented reality with varying materials on mobile devices. In *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, pages 1349–1362. (page 18)

[113] Rosenberg, L. B. (1992). The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments. Technical report, Stanford Univ CA Center for Design Research. (page 3)

[114] Saakes, D., Choudhary, V., Sakamoto, D., Inami, M., and Lagarashi, T. (2013). A teleoperating interface for ground vehicles using autonomous flying cameras. In *Proc. ICAT*, pages 13–19. (page 22)

[115] Sandor, C., Cunningham, A., Dey, A., and Mattila, V.-V. (2010). An augmented reality x-ray system based on visual saliency. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 27–36. IEEE. (page 24)

[116] Sandor, C., Cunningham, A., Eck, U., Urquhart, D., Jarvis, G., Dey, A., Barbier, S., Marner, M. R., and Rhee, S. (2009). Egocentric space-distorting visualizations for rapid environment exploration in mobile mixed reality. In *Proc. ISMAR*, pages 211–212. (page 24)

[117] Schmalstieg, D. and Höllerer, T. (2016). *Augmented Reality - Principles and Practice*. Addison-Wesley Professional. (page 5)

[118] Schmuck, P. and Chli, M. (2017). Multi-UAV collaborative monocular SLAM. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3863–3870. IEEE. (page 14)

[119] Schöps, T., Oswald, M. R., Speciale, P., Yang, S., and Pollefeys, M. (2017). Real-time view correction for mobile devices. *IEEE Trans. on Visualization and Computer Graphics*, 23(11):2455–2462. (page 59)

[120] Scott, W. R., Roth, G., and Rivest, J.-F. (2003). View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv.*, 35(1):64–96. (page 3, 8)

[121] Shen, Y., Lu, F., Cao, X., and Foroosh, H. (2006). Video completion for perspective camera under constrained motion. In *Proc. Int. Conf. on Pattern Recognition*, pages 63–66. (page 17)

[122] Siltanen, S. (2006). Texture generation over the marker area. In *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, pages 253–254. (page 16, 54, 64)

[123] Siltanen, S. (2015). Diminished reality for augmented reality interior design. *The Visual Computer*, 33:193–208. (page 16, 18, 50, 55, 64)

[124] Steinbrücker, F., Sturm, J., and Cremers, D. (2011). Real-time visual odometry from dense rgb-d images. In *ICCV Workshops*, pages 719–722. IEEE Computer Society. (page 65)

[125] Stoakley, R., Conway, M. J., and Pausch, R. (1995). Virtual reality on a wim: Interactive worlds in miniature. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 265–272, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. (page 102)

[126] Sugimoto, M., Kagotani, G., Nii, H., Shiroma, N., Matsuno, F., and Inami, M. (2005). Time follower's vision: a teleoperation interface with past images. *IEEE Computer Graphics and Applications*, 25(1):54–63. (page 22)

[127] Tatzgern, M., Grasset, R., Kalkofen, D., and Schmalstieg, D. (2014). Transitional AR Navigation for Live Captured Scenes. In *Proc. IEEE VR*. (page 84)

[128] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Proc. Int. Conf. on Computer Vision*, pages 839–846. (page 54)

[129] Untzelmann, O., Sattler, T., Middelberg, S., and Kobbelt, L. (2013). A scalable collaborative online system for city reconstruction. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 644–651. (page 14)

[130] Valentin, J., Kowdle, A., Barron, J. T., Wadhwa, N., Dzitsiuk, M., Schoenberg, M., Verma, V., Csaszar, A., Turner, E., Dryanovski, I., Afonso, J., Pascoal, J., Tsotsos, K., Leung, M., Schmidt, M., Guleryuz, O., Khamis, S., Tankovitch, V., Fanello, S., Izadi, S., and Rhemann, C. (2018). Depth from motion for smartphone ar. *ACM Trans. on Graphics*, 37(6):193:1–193:19. (page 18)

[131] Vázquez, P.-P., Feixas, M., Sbert, M., and Heidrich, W. (2004). Automatic view selection using viewpoint entropy and its application to image-based modelling. *Computer Graphics Forum*, 22(4):689–700. (page 8, 15)

[132] Waechter, M., Beljan, M., Fuhrmann, S., Moehrle, N., Kopf, J., and Goesele, M. (2017). Virtual rephotography: Novel view prediction error for 3d reconstruction. *ACM Transactions on Graphics (TOG)*, 36(1):1–11. (page 37)

[133] Waechter, M., Moehrle, N., and Goesele, M. (2014). Let there be color! Large-scale texturing of 3D reconstructions. In *ECCV*, pages 836–850. (page 5, 7, 14)

[134] Wang, J. Y. A. and Adelson, E. H. (1994). Representing moving images with layers. *IEEE Trans. on Image Processing*, 3(5):625–638. (page 17)

[135] Wang, Y., Krum, D. M., Coelho, E. M., and Bowman, D. A. (2007). Contextualized videos: Combining videos with environment models to support situational understanding. *IEEE TVCG*, 13(6):1568–1575. (page 23)

[136] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612. (page 37)

[137] Weilharter, R. J., Schenk, F., and Fraundorfer, F. (2018). Globally consistent dense real-time 3D reconstruction from RGBD data. In *OAGM Workshop*. (page 37)

[138] Yeh, R. A., Chen, C., Lim, T. Y., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. (2017). Semantic image inpainting with deep generative models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 6882–6890. (page 19)

[139] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., and Huang, T. S. (2018). Generative image inpainting with contextual attention. In *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. (page 19)

[140] Zhang, E., Cohen, M. F., and Curless, B. (2016). Emptying, Refurnishing, and Relighting Indoor Spaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2016)*, 35(6). (page 5, 15)

[141] Zhou, Q.-y. and Koltun, V. (2014). Color map optimization for 3D reconstruction with consumer depth cameras. *ACM Transactions on Graphics*, 33(4):1–10. (page 14)

[142] Zhu, C., Yu, L., and Xiong, Z. (2018). A Noncoverage Field Model for Improving the Rendering Quality of Virtual Views. *IEEE Trans. Multimed*, 20(3):738–753. (page 15)

[143] Zokai, S., Esteve, J., Genc, Y., and Navab, N. (2003). Multiview paraperspective projection model for diminished reality. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, pages 217–226. IEEE. (page 17)

[144] Zollmann, S., Hoppe, C., Langlotz, T., and Reitmayr, G. (2014). FlyAR: AR supported micro aerial vehicle navigation. *TVCG*, 20(4):560–568. (page 22)