# Viewpoint-Free Photography for Virtual Reality

*Peter Hedman*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

**Doctor of Philosophy**

of

**University College London**.

Department of Computer Science

University College London

July 15, 2019

# Abstract

Viewpoint-free photography, i.e., interactively controlling the viewpoint of a photograph after capture, is a standing challenge. In this thesis, we investigate algorithms to enable viewpoint-free photography for virtual reality (VR) from casual capture, i.e., from footage easily captured with consumer cameras.

We build on an extensive body of work in image-based rendering (IBR). Given images of an object or scene, IBR methods aim to predict the appearance of an image taken from a novel perspective. Most IBR methods focus on full or near-interpolation, where the output viewpoints either lie directly between captured images, or nearby. These methods are not suitable for VR, where the user has significant range of motion and can look in all directions. Thus, it is essential to create viewpoint-free photos with a wide field-of-view and sufficient positional freedom to cover the range of motion a user might experience in VR.

We focus on two VR experiences:

1) Seated VR experiences, where the user can lean in different directions. This simplifies the problem, as the scene is only observed from a small range of viewpoints. Thus, we focus on easy capture, showing how to turn panorama-style capture into 3D photos, a simple representation for viewpoint-free photos, and also how to speed up processing so users can see the final result on-site.

2) Room-scale VR experiences, where the user can explore vastly different perspectives. This is challenging: More input footage is needed, maintaining real-time display rates becomes difficult, view-dependent appearance and object backsides need to be modelled, all while preventing noticeable mistakes. We address these challenges by: (1) creating refined geometry for each input photograph, (2) using a fast tiled rendering algorithm to achieve real-time display rates, and (3) using a convolutional neural network to hide visual mistakes during compositing.

Overall, we provide evidence that viewpoint-free photography is feasible from casual capture. We thoroughly compare with the state-of-the-art, showing that our methods achieve both a numerical improvement and a clear increase in visual quality for both seated and room-scale VR experiences.

# Acknowledgements

I would like to thank my supervisor Gabriel Brostow for his support and relentless enthusiasm throughout my PhD — many successful projects would have fizzled out without him.

Many thanks to Johannes Kopf who brings his superpowers to any project he touches. The projects in Chapters 3-4 would never have happened without his energy, technical advice, creative ideas and stellar salesmanship.

Thanks to Rick Szeliski for taking me in during my lonely Christmas in Seattle, and for being a role model: one does not have to sacrifice the other sides of life to be a successful researcher.

Thanks to Clément Godard for his incredible attention to detail, many bugs have been spotted and fixed because of him. I am also grateful for his honesty in shooting down many of my unrealistic and whimsical ideas — this saved me from countless days of wasted effort.

Thanks to George Drettakis, whose burning passion for image-based rendering brought me into this field, for finding the elegance in our projects.

Thanks also to Tobias Ritschel for his invaluable help on high-performance rendering algorithms, without his pragmatism Chapter 5 would never have made it.

I am incredibly thankful to everyone who were in the trenches with me near deadlines. To Julien Philip, True Price and Suhib Alsisan who all stayed up late in a race against time to put together beautiful research papers.

I am also hugely grateful that Moos Hueting, Tara Ganepola, Corneliu Ilisescu and Aron Monszpart selflessly came to the rescue during the darkest hours of my first deadline at UCL. I am touched by their incredible kindness and will never forget the evening when everyone came together to help save our sinking ship.

Thanks to Jaakko Lehtinen, Tero Karras, Samuli Laine and Timo Aila for inspiring and guiding me when I was a young over-eager programmer to pursue research in computer graphics.

To everyone I lived with in London: Lotta Buxton, Gerry Buxton, Valter Holmström and Carolina Galvão. Thanks for putting up with my crazy schedule and for keeping my mind off work, either through deep conversations about life, or by simply watching mindless TV shows together.

Finally, I owe this thesis to caffeinated beverages, my parents, and Aleksander Hołyński. Without them I would never have been able to put it together.

# Impact Statement

In this thesis, we present four novel approaches to viewpoint-free photography. Each approach was presented at either SIGGRAPH or SIGGRAPH Asia, the premier venues for scholarly work in the field of computer graphics. To encourage future academic work, results and datasets have been released for the methods in Chapters 3, 5 and 6.

The Casual 3D Photography method presented in Chapter 3 was developed during an internship at Facebook, which later acquired a patent for the technology [1]. The method was also presented by Facebook CEO Mark Zuckerberg in his keynote at F8, the yearly Facebook conference for developers[1].

The Instant 3D Photography approach in Chapter 4 inspired the new 3D photo feature available today in the Facebook application. It also was featured in a TechCrunch article [2] and was presented in the keynote at the 2018 F8 conference[2].

The Scalable Free-Viewpoint Image-Based Rendering method presented in Chapter 5 was developed for the EU funded CR-Play project [3], which investigated how viewpoint-free photography methods can be used to make asset creation easier for video game developers. This method was later used as a core technology for an early-stage startup which continued the work of the EU project.

Finally, the technologies developed for Chapter 6 are used as the basis for future research in the GraphDeco group[3] at INRIA Sophia-Antipolis Méditerranée Research Centre.

[1]`https://developers.facebook.com/videos/f8-2017/f8-2017-keynote` — at 11:45.
[2]`https://developers.facebook.com/videos/f8-2018/f8-2018-day-2-keynote` – at 1:08:45.
[3]`https://team.inria.fr/graphdeco`

# Publications

**Chapter 3:** Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3D photography. *ACM Trans. Graph.*, 36(6):234:1–234:15, 2017

Richard Szeliski offered helpful insight on multi-view stereo, and helped design the color-and-depth panorama stitching approach. Suhib Alsisan built a server implementation of the pipeline and helped perform evaluation. Johannes Kopf designed and implemented the two-layer fusion approach in Chapter 3.2.4.3 (Section 4.4.3 in the paper), and the normal map syntesis approach (Sec 5 in the paper).

**Chapter 4:** Peter Hedman and Johannes Kopf. Instant 3D photography. *ACM Trans. Graph.*, 37(4):101:1–101:11, 2018

Johannes Kopf designed and implemented the multi-layer processing in Chapter 4.2.4 (Section 4.4 in the paper). Suhib Alsisan implemented the capture application.

**Chapter 5:** Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM Trans. Graph.*, 35(6):231:1–231:11, 2016

Tobias Ritschel offered helpful insight on high-performance rendering. George Drettakis helped improve the quality of the rendering algorithm. Gabriel Brostow provided insight for depth-sensor reconstruction, and captured half of the data.

**Chapters 5-6:** Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 37(6):257:1–257:15, 2018

Julien Philip designed and implemented the meshing approach in Chapter 5.2.3 (Section 4.2 in the paper). True Price wrote the TensorFlow-OpenGL interoperability layer and helped perform evaluation. Jan-Michael Frahm provided insight on 3D reconstruction. George Drettakis positioned the paper w.r.t ṁodern image-based rendering approaches. Gabriel Brostow helped with formulating the blending network and participated in data capture.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

**Figure 1.1:** Over time, we have strived to realistically capture and immersively convey the visual sensation of a place, starting with paintings like the Arnolfini Portrait by Jan van Eyck (left). Modern cameras made it easy for anyone to capture realistic 2D images (center). In this thesis, we look beyond 2D images and develop methods to easily capture *viewpoint-free photos* that can be immersively experienced in modern VR headsets (right).

Imagine if you could visually capture any place, in a way that allows anyone to immersively re-experience the sensation of being there. This goal lies at the intersection of art and science, and is something we have been working toward for centuries. Already in the 15th century, realistic paintings — such as the Arnolfini Portrait by Jan van Eyck (Figure 1.1) — painstakingly captured effects such as perspective, illumination and even complex reflections off mirrors. Later, with the advent of photography in the 19th century, it became possible to easily create two-dimensional (2D) images with this degree of realism. Since then, we captured and preserved a much greater amount of realistic 2D content, giving us a rich visual history of the 20th century. Ultimately, the democratization of photography gave this type of content social and personal significance. Today, you can capture photos of places you visit, share them with friends and family, so they feel more connected to you, or preserve your personally treasured places forever as visual memories.

However, photos only capture a single perspective and cannot fully convey the experience of a place. Consequently, more immersive media experiences have since been created. In general, each new media technology is first introduced using content that has been painstakingly captured by experts. The technology that makes it easy and affordable for the general public to capture realistic content often comes much later. For example, the professional motion picture industry dates back to the late 19th century, but motion pictures only became easy and affordable to capture with the introduction of low-cost 8mm film cameras in the 1930s. In the 1990s early virtual-reality (VR) headsets like the Forte VFX1 increased immersion by displaying different perspectives for each eye (stereoscopic 3D) and enabling the user to rotate in-place to interactively explore the scene [8]. Content for this type of media only recently became easy to capture once modern stereo panorama stitching techniques [9, 10] made it into consumer cameras [11] and smartphone applications [12].

Modern VR headsets with positional tracking, such as the Oculus Rift and HTC Vive, enable more immersive experiences beyond simply rotating in place. For example, *seated VR* is where the user can lean in different directions and peek behind corners, and *room-scale VR* is where the user can freely walk around the room and view the scene from vastly different perspectives. However, there isn't yet a technology to easily capture real places for these experiences. Such technology would allow everyone to experience hard to reach places, such as the wreck of the Titanic or the peaks of Mount Everest. It also has commercial applications, for example in real estate, where it is useful to present property on sale in realistic 3D [13], or for video game development where 3D asset creation is costly [3]. Finally, this technology has social and personal significance: it would enable people to capture places of personal or cultural importance, and either share these with friends, or preserve them forever as digital memories.

## 1.1 Goals

In this thesis, we address the problem of easily capturing real places for realistic display in VR headsets. We call this type of capture *viewpoint-free photography*, i. e., you capture a real place without fixing the viewpoint in advance, and strive toward the following goals:

1. ease of capture,
2. sufficient range of motion,
3. high quality (realism), and
4. real-time display rates.

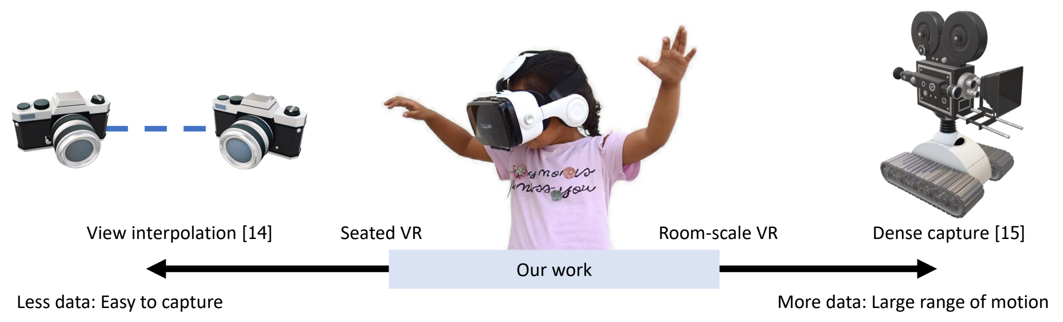


**Figure 1.2:** The amount of input data causes a trade-off between ease of capture and range of motion. Naturally, you cannot display parts of the scene that were not captured by the input footage. Earlier work on visual reconstruction and image-based rendering tends to explore both extremes of this spectrum, e. g., by strictly interpolating between a small number of input images [14], or by exhaustively capturing a dense grid of viewpoints in the scene using a robot [15]. We explore the middle of this spectrum, determined by the range of motion needed for seated VR experiences and room-scale VR experiences.

Specifically, we only use off-the-shelf hardware, and look for approaches that can capture a real place in a short amount of time (1). The resulting viewpoint-free photos should allow the user sufficient range of motion for either a seated VR or a room-scale VR experience (2) without noticing mistakes whose appearance in the photos significantly differ from reality (3). Finally, to allow interactive exploration in VR, we investigate approaches that can quickly display 2D images at novel viewpoints at real-time rates of 30 images per second or more (4). This leads us to the following hypothesis:

> *We can create viewpoint-free photos, suitable for viewing in virtual reality, from casually captured footage, i. e., footage that can be captured in less than 30 minutes using an off-the-shelf camera.*

To achieve this, we look for an appropriate capture method, representation, and reconstruction algorithm for:

- the range of motion needed for seated VR experiences, and
- the much larger range of motion supported in untethered, room-scale VR experiences.

## 1.2 Context

In this thesis, we make heavy use of recent advances in the closely related, but subtly different, field of 3D reconstruction from images (or *multi-view stereo, MVS*) [16, 17, 18]. While our focus is on accurately capturing the appearance of the scene, multi-view stereo is less concerned with appearance and works towards accurate 3D geometry. Instead, our work

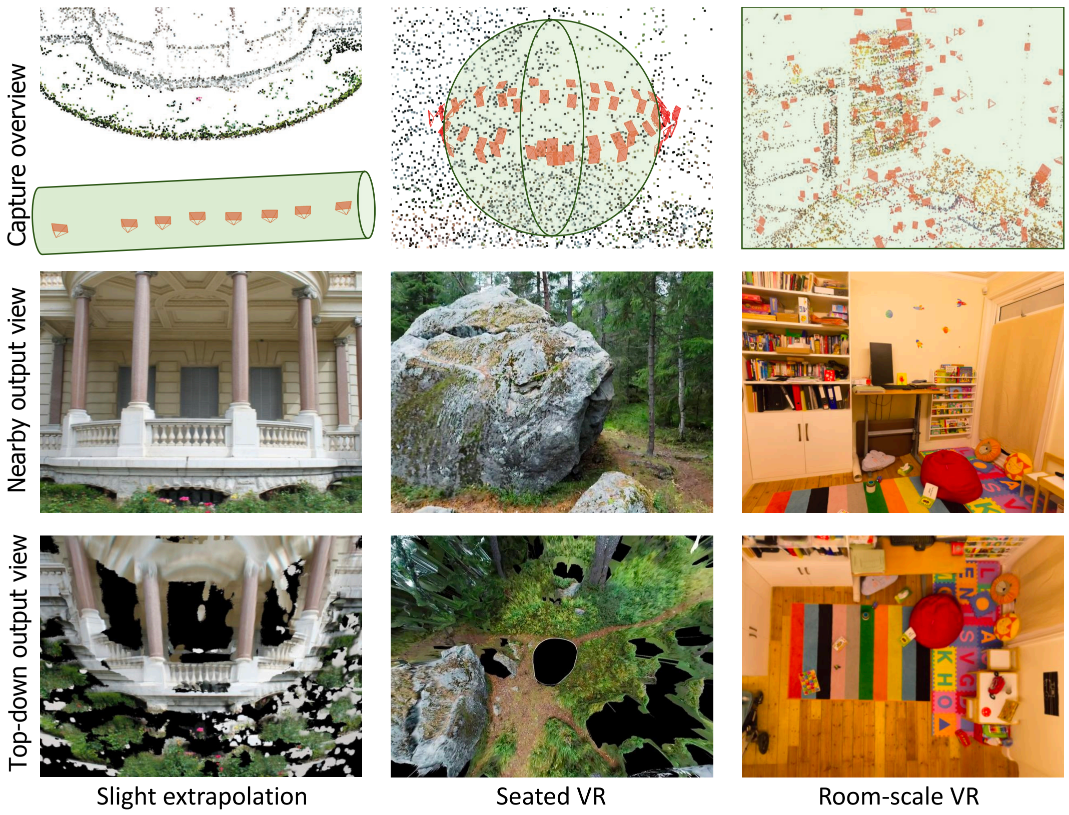


**Figure 1.3: Left-to-right:** Image-based rendering for slight view extrapolation (Soft 3D [24]), viewpoint-free photography for seated VR (Chapters 3-4), viewpoint-free photography for room-scale VR (Chapters 5-6). The top row shows the input photo locations in red and the range of motion enabled by each method is visualized in green. Note that the left column is not directly suitable for VR experiences — it does not capture the entire scene, as the input photos all look towards the same direction. All methods look great when staying close to the input photos (middle row), but only our approach for room-scale VR is able to maintain high-quality results with a difficult top-down view, which is far away from the input photos (bottom row).

sits in the space of image-based rendering (IBR), which covers all methods that synthesize images from a collection of input photos in a scene.

As Figure 1.2 shows, in IBR there is a trade-off between ease of capture (goal 1) and range of motion (goal 2). Traditionally, IBR techniques have been clustered at opposites ends of this spectrum. For example, early IBR approaches [19, 20] can be extended to deliver a large range of motion for room-scale scenes [15], but only with robot-assisted capture rigs to automate the tedious and time-consuming process of exhaustively capturing a dense set of viewpoints in the scene. At the other end of the spectrum, the emphasis is on easy capture, where early work focused on strict interpolation between the input viewpoints [14, 21, 22]. Recent approaches have shown results with view extrapolation [23, 24], allowing for a slight increase in range of motion compared to strict interpolation.

Our work is unique because we target easy-to-capture VR experiences, which lie in the middle of the spectrum. In particular, the rotational range of motion in VR headsets is immense: the user can rotate freely and look in all directions, making it essential to create viewpoint-free photos with a wide field-of-view (FOV). As illustrated in Figure 1.3 (left), earlier methods for easy-to-capture IBR are generally limited to narrow FOV experiences with a small rotational range of motion.

In Chapters 3-4, we target seated VR experiences with a limited range of motion, see Figure 1.3 (middle). This simplifies the representation and reconstruction problem, as the scene is only ever observed from a small range of viewpoints. For example, the user only ever sees the front facing side of objects in the scene and the view-dependent appearance of materials becomes less important to preserve. We thus focus on ease of capture (goal 1), showing how to turn panorama-style capture into *3D photos*, a simple representation for viewpoint-free photos (Chapter 3), and also how to speed up processing so users can see the final result and make adjustments on-site (Chapter 4).

In Chapters 5-6, we target room-scale VR experiences where the user is free to explore the scene without constraints, see Figure 1.3 (right). In this setting, we have to capture more data from different viewpoints in the scene, so instead of focusing on ease-of-capture, we place our effort on realism, quality (goal 3) and real-time display rates (goal 4). In particular, maintaining real-time display rates becomes difficult with more data, we also have to pay attention to view-dependent appearance, build a representation for the backsides of objects, and also make sure that there are no noticeable mistakes in the locations the user might visit.

## 1.3 Contributions

We show how to easily capture and create viewpoint-free photos with a wide field-of-view and large rotational range of motion using just an off-the-shelf camera.

For viewpoint-free photography targeting seated VR experiences, our contributions are:

- A novel parallax-tolerant color-and-depth stitching approach which creates 3D photos from panorama-style capture.
- A fast novel method to jointly align and correct for geometric deformations in color-and-depth images, making it possible to rapidly create 3D photos from panorama-style capture with dual-camera mobile phones.

For viewpoint-free photography targeting room-scale VR experiences, our contributions are:

- A per-input-photo geometry refinement approach for viewpoint-free photos that preserves view-dependent appearance and with well-aligned occlusion edges.
- A fast novel, tiled rendering algorithm for free-viewpoint photos based on a large amount of per-input-photo geometry.
- A novel deep convolutional neural network approach to suppress noticeable mistakes in viewpoint-free photos.

## 1.4 Structure

The rest of this thesis is organized as follows:

- In Chapter 2, we discuss previous work on 3D reconstruction, computational photography, image-based rendering that is relevant to the methods presented in this thesis.
- In Chapter 3, we develop a viewpoint-free photography approach for seated VR experiences. We present a novel color-and-depth stitching algorithm that creates viewpoint-free photos from panorama-style input footage, which is easy to capture.
- In Chapter 4, we continue our work on turning easily captured panorama footage into viewpoint-free photos for seated VR experiences, placing our focus on fast processing, so users can see the final result and on-site. We present a novel, fast method to jointly align and correct deformations in color-and-depth images, making it possible to rapidly create viewpoint-free photos from color-and-depth panorama footage, captured with e.g. a dual camera phone.
- In Chapter 5, we shift our attention towards room-scale VR experiences. We propose a system to create and render viewpoint-free photos for this experience from footage captured in less than 30 minutes with an off-the-shelf camera. In particular, we achieve high-quality results with a per-input-photo geometry refinement approach and maintain real-time display rates with a novel, fast tiled rendering algorithm.
- In Chapter 6, we continue our work on room-scale VR experiences, and focus on maintaining consistent high-quality with a novel approach to suppressing mistakes

in viewpoint-free photos. We replace the blending step from the system presented in Chapter 5 with a convolutional neural network that has been trained to hide mistakes.

- Finally, in Chapter 7, we conclude with lessons learned, limitations, and exciting avenues for future work.

Chapter 2

# Background

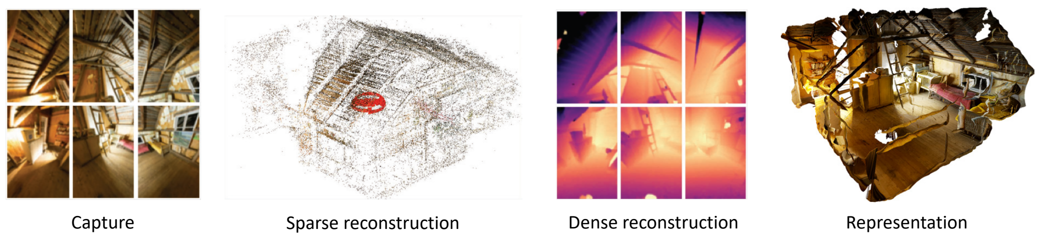


**Figure 2.1:** Generic pipeline for viewpoint-free photography.

Our work on viewpoint-free photography builds on a long tradition of 3D reconstruction from images [25] and image-based rendering (IBR) [26]. In this chapter, we discuss technical preliminaries and provide an overview of related work in 3D reconstruction and IBR. Figure 2.1 depicts a generic pipeline for viewpoint-free photography. In general, most approaches to viewpoint-free photography and IBR can be broken down into four steps:

**Capture:** How is the input footage obtained? Does the method need a custom-built capture rig, or is hand-held footage from an off-the-shelf camera sufficient?

**Sparse reconstruction:** This steps resolves how the captured images relate to each other in space. Most commonly, this step estimates the 3D pose for each input photo.

**Dense reconstruction:** Once the camera poses are known, most approaches compute an estimate of the 3D geometry in the scene.

**Representation and rendering:** The final step is a *representation* for the viewpoint-free photo — i. e., how the estimated geometry is combined with color data — coupled with a *rendering algorithm* which uses the representation to generate output images.

In the following we will describe each stage in more depth.

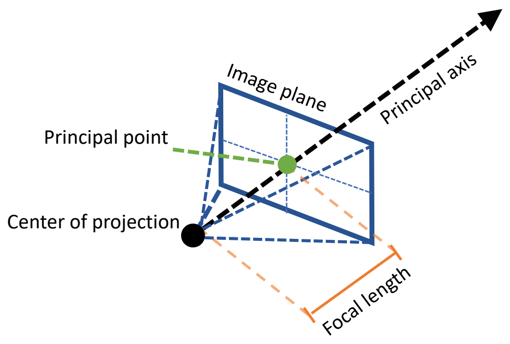


**Figure 2.2:** The pinhole camera model and associated concepts.

## 2.1 Capture and sparse reconstruction

Capture is an important part of any viewpoint-free photography pipeline. Not only does the capture strategy determine the usability of the final system, but it also has a large impact on the technical aspects for the rest of the pipeline stages. Some issues and visual mistakes can often be completely avoided with a more exhaustive capture process. After capture, the next crucial step is to recover the camera location and orientation for each image that was captured. We call this process *sparse reconstruction* as it often simultaneously recovers a sparse 3D model of the scene.

In this section, we discuss different strategies for capture and sparse reconstruction. For an in-depth treatment on the technical details, we refer the interested reader to the in-depth treatment by Hartley and Zisserman [27].

### 2.1.1 Image formation with the pinhole camera model

We begin with a simple mathematical description of capturing a photograph. For this purpose, we use the *pinhole camera* model illustrated in Figure 2.2. That is, instead of using a lens system, we gather light onto the camera sensor through an infinitesimal hole. It is worth noting that this does not account for distortions caused by lens systems in real cameras — refer to [28, Chapter 2.1.6] for mathematical models of lens distortion.

#### 2.1.1.1 Transformation

As shown in Fig 2.3, the first step of the image formation process is to transform the geometry in the scene to a frame of reference centered around our camera. 3D transformations, or general transformations between 3D points, can be ordered into several classes, which include translations, rotations, and projections [27, Section 2.4]. Generalized 3D transformations (at least those which are useful for our purposes), can be represented as $4 \times 4$ matrices. Given

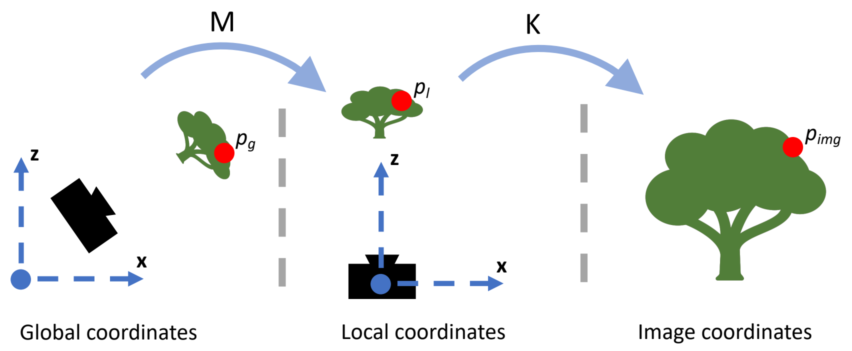


**Figure 2.3:** The geometry of image formation, left-to-right: The extrinsics $\mathbf{M}$ transforms a point $\mathbf{p}_g$ from a global frame of reference to a point $\mathbf{p}_l$ in the local coordinate space centered on the camera. The intrinsics $\mathbf{K}$ projects points into image space, mapping $\mathbf{p}_l$ onto $\mathbf{p}_{img}$.

a homogeneous 3D point $\mathbf{p}_g = (x_g, y_g, z_g, 1)$ in an common, global frame of reference, we apply the extrinsics matrix $\mathbf{M}$, a $4 \times 4$ transformation matrix, which transforms the point into the coordinate frame of our camera:

$$\mathbf{p}_l = \mathbf{M}\mathbf{p}_g. \tag{2.1}$$

After alignment, the point $\mathbf{p}_l$ is expressed in local coordinates, where the origin is at the *center of projection* of the camera, and the z-axis aligns with the camera's *principal axis*. The extrinsics matrix $M$ is composed of a $3 \times 3$ rotation $\mathbf{R}$ and a $3 \times 1$ translation $\mathbf{t}$.

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tag{2.2}$$

### 2.1.1.2 Projection

Once the geometry has been transformed to the frame of reference centered around our camera (often referred to as the *local* coordinate system), we might want to visualize or render the 3D geometry from the point of view of the camera. In order to do so, it is necessary to find the corresponding 2D coordinates for each of the 3D points. This process is called *projection*, and is done using a $3 \times 4$ projection matrix, $K$:

$$\mathbf{p}_{2D} = \mathbf{K}\mathbf{p}_l, \tag{2.3}$$

where $\mathbf{p}_{2D} = (x_{2D}, y_{2D}, z_{2D})$ is still expressed in homogeneous coordinates. For the most common form of projection, *perspective projection*, a final divide is necessary to obtain image-space 2D coordinates:

$$\mathbf{p}_{img} = n(\mathbf{p}_{2D}) = \left(\frac{x_{2D}}{z_{2D}}, \frac{y_{2D}}{z_{2D}}\right). \tag{2.4}$$

This division fuction $n$ is responsible for the perspective effect, where far-away objects appear smaller. For our pinhole camera model, the projection matrix $K$ has the format:

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{2.5}$$

where $f$ is the focal length of the lens and $p_x$ as well as $p_y$ are the coordinates of the *principal point*, which is the image-space location of the principal axis, i. e., the $z$-axis in the local coordinate space of the camera.

### 2.1.1.3 Depth and disparity

A fundamental concept in the field of scene reconstruction is the notion of depth. We measure depth of a 3D point by measuring its distance to the plane defined by camera sensor. This is equivalent to the $z$ coordinate of the point in the local coordinate system of our camera.

When reasoning about the image-space motion of 3D points, it is common to reason about their inverse depths, or *disparities* $d = z^{-1}$. This is mitigates numerical precision issues for scenes with large range of depths, e. g., outdoor scenes where the horizon is visible.

### 2.1.1.4 Radiometric image formation

Using the tools above, we are able to project a 3D point into a 2D image. This determines the locations on the camera sensor which will observe certain parts of the scene, but does not yet allow us to predict the appearance of the scene in the image. For this, we need a *radiometric* model for image formation, i. e., how the camera sensor responds to different wavelengths and intensities of light. For the purposes of this thesis, we use a somewhat simplified model of a camera sensor which ignores subpixel placement and demosaicking algorithms, please refer to the treatment by Szeliski [28, Chapter 2.2] for a more detailed description.

We consider the camera sensor to consist of a grid of rectangular *pixels*. Each pixel measures how much light it receives in three separate frequency bands, corresponding to the primary colors red, green and blue. For each primary color, we model the response $r = f(p)$ measured by each pixel as of how many photons $p$ that land on the pixel. The *response curve*

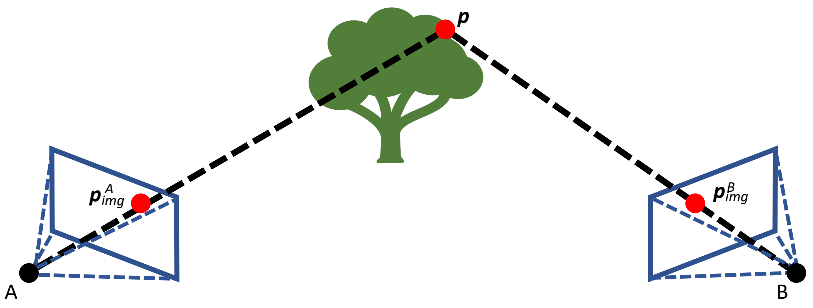


**Figure 2.4:** Two-view geometry. A 3D point **p** and its 2D projections into two images A and B.

$f$ of the sensor, is a non-linear function which varies from camera to camera. Naturally, the overall brightness of the image can be adjusted by changing the shutter speed of the camera, i. e., the duration for which the sensor will be exposed to light.

In general, we cannot trust the pixel response for a given point in the scene to be the same across different photographs — even if the camera and exposure time remain unchanged. This is because each pixel produces noisy measurements, an effect which is particularly visible in low-light conditions. Furthermore, camera lenses tend to gather more light towards the center of the sensor, causing edge darkening or *vignetting*. Finally, while ideal *diffuse* materials would reflect the same amount of light in all directions, most materials have view-dependent appearance due to effects such as highlights and Fresnel reflection.

### 2.1.2 Multi-view capture

To effectively reason about the 3D structure of a scene, we often need to capture footage from several different viewpoints. Here, we discuss mathematical preliminaries and useful concepts for *multi-view capture*, i. e., when several photos have been taken in the same scene. We also examine different capture strategies used by existing viewpoint-free photography pipelines, to see how well they align with the goals we defined in Section 1.

#### 2.1.2.1 Reprojection and triangulation

Consider the scene pictured in Figure 2.4, containing two images A and B, both with known extrinsics ($\mathbf{M}_A = [\mathbf{R}_A\ \mathbf{t}_A], \mathbf{M}_B = [\mathbf{R}_B\ \mathbf{t}_B]$) and intrinsics ($\mathbf{K}_A, \mathbf{K}_B$) matrices. Using this information, we can relate the 2D coordinates in both images to a 3D point in the scene.

**Reprojection.** Assume that we have been able to estimate the depth $z_A$ at a single pixel location $\mathbf{p}^{A}_{img}$ in image A. With this information we determine the 2D coordinates $\mathbf{p}^{B}_{img}$ of

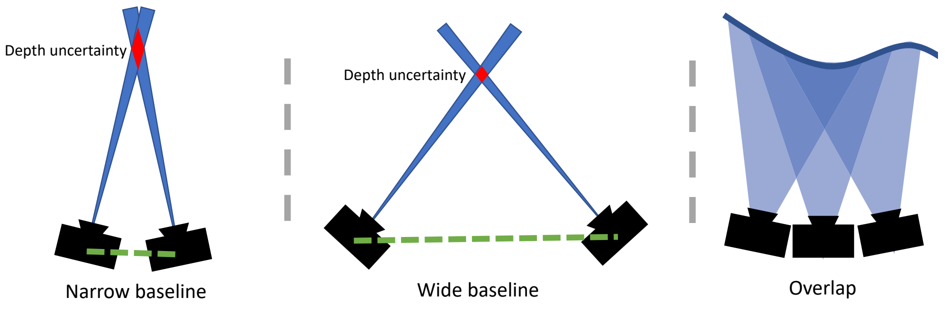


**Figure 2.5:** Relevant concepts for multi-view capture. **Left:** With a narrow baseline (dotted green line) between cameras, the triangulation angle between a pair of corresponding pixels is often small, leading to a large uncertainty in the estimated depth. **Middle:** With a larger baseline, the uncertainty region shrinks. **Right:** Overlap is a convenient measure for capture density. Here, the central part of the scene has three-way overlap.

this pixel as seen in image B through *reprojection*, i. e.,

$$\mathbf{p}^B_{img} = P_{A\to B}(\mathbf{p}^A_{img}) = n(\mathbf{K}_B(\mathbf{R}_B\mathbf{R}_A^{-1}(z_A\mathbf{K'}_A^{-1}\hat{\mathbf{p}}^A_{img} - \mathbf{t}_A) + \mathbf{t}_B)), \tag{2.6}$$

where $\hat{\mathbf{p}}^A_{img}$ is the homogeneous-augmented version of $\mathbf{p}^A_{img}$, $\mathbf{K'}_A^{-1}$ is the inverse of the leftmost $3\times 3$ submatrix in $\mathbf{K}_A$, and the function $n$ performs the perspective divide (see Equation 2.4).

**Triangulation.** Now assume, that instead of knowing the depth of a pixel, we have been able to establish a *correspondence* between the two images, i. e., we have found a 3D point that is visible in both cameras. It is important to note that the 3D location $\mathbf{p}$ of this point is unknown, and that the 2D coordinates $(\mathbf{p}^A_{img}, \mathbf{p}^B_{img})$ have been estimated with a certain degree of error in both images, due to e. g., sensor noise and the resolution of the pixel grid.

It is possible to estimate the 3D location of the point by through *triangulation*, by finding a 3D point $\mathbf{p}'$ which is close to $\mathbf{p}^A_{img}$ when projected into image A, and close to $\mathbf{p}^B_{img}$ when projected into image B. Formally, this amounts to minimizing

$$|\mathbf{p}^A_{img} - n(\mathbf{K}_A\mathbf{M}_A\mathbf{p}')|^2 + |\mathbf{p}^B_{img} - n(\mathbf{K}_B\mathbf{M}_B\mathbf{p}')|^2, \tag{2.7}$$

although commonly an approximation is made, which measures the distance using homogeneous coordinates. This is the direct linear transform method (DLT) described by Hartley and Zisserman [27, Chapter 12.2], which generalizes to an arbitrary number of cameras.

### 2.1.3 Capture strategies

To better analyse and compare different capture strategies for viewpoint-free photography, we introduce the concepts illustrated in Figure 2.5: *Triangulation angle*, *baseline*, and *overlap*:

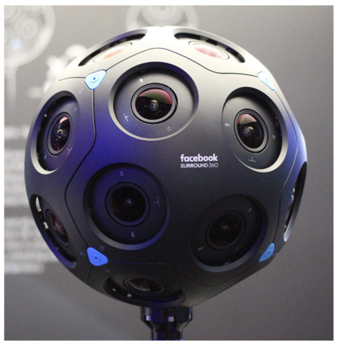


**Figure 2.6:** Capture methods. **Left:** Rig capture with the Facebook Surround x24 360 (Photo by Maurizio Pesce, CC BY 2.0[1]). **Right:** Unstructured capture with a hand-held camera.

**Triangulation angle.** This is the angle between the two rays cast from a pair of corresponding pixels in a pair of cameras. As the figure demonstrates, narrow triangulation angles often lead to large inaccuracies when estimating depth. This is due to small, but inevitable errors in the 2D locations that have been estimated for the feature correspondences. These pixel errors result in inaccuracies in the depth estimates during triangulation, and narrow triangulation angles further exaggerate this effect.

**Baseline.** The baseline is the distance between the centers of projection for two cameras in the scene. A larger baseline, generally leads to wider triangulation angles and better accuracy for triangulated depths. However, with wide baselines it is often more difficult to find corresponding pairs of pixels in the first place.

**Overlap.** Overlap is a useful measure for capture density in the scene. It refers to the number of input images which observe a particular part of a scene, or the average fraction of an image which is shared by its nearest neighbors. Typically, a higher overlap is desired, as from more input cameras lead to reliable depth estimates.

#### 2.1.3.1 Rig capture

One of the most reliable forms of multi-view capture is known as rig capture, seen in Figure 2.6 (left). Rig capture refers to the use of a *rig*, or specially designed multi-camera setup, in which the cameras and their relative positions and orientations are fixed. In this way, these systems can be *precalibrated* (their intrinsics **K**, extrinsics **M**, and radiometric properties estimated), so that when footage is captured, the camera parameters and positions

[1]`https://www.flickr.com/photos/pestoverde/34843129056`
`https://creativecommons.org/licenses/by/2.0`

do not need to be estimated, greatly reducing the likelihood of error in reconstruction. These systems are most often used for professional applications, like film production.

Indeed, it is possible to create very high quality viewpoint-free photographs by exhaustively capturing footage with a motorized-multi camera rig [29]. This is well suited for seated VR experiences with complete 360° coverage, but a limited range of motion. Alternatively, viewpoint-free *video* capture is possible with rigs where the cameras do not move, albeit with lower overlap in captured footage [30, 31]. This makes 3D reconstruction less reliable, further reducing the feasible range of motion. It is possible to increase the range of motion using a multi-camera capture *studio* instead of just a single rig [32, 33, 34, 35], making viewpoint-free video for room-scale VR experiences plausible.

While incredibly robust, these rigs and studios do not facilitate our goal of easy capture, as they are quite expensive, requiring well-synchronized high-end cameras mounted using custom hardware. In addition, they are usually quite cumbersome, and not the type of device an average consumer is likely to buy or use.

#### 2.1.3.2 Unstructured capture

On the other end of the spectrum is unstructured capture (Figure 2.6, right). In contrast to rig capture, unstructured capture strategies allow the input cameras to be at arbitrary positions. Consequently, these approaches need to estimate the intrinsic parameters, relative orientations, and radiometric properties of the cameras.

There is a large body of working demonstrating that unstructured capture can be used to create viewpoint-free photos [36, 22, 23], albeit these methods often ingest footage with close to linear camera motion and almost no rotation. This limits the permissible range of motion in the resulting viewpoint-free photographs, making this linear capture style unsuitable for VR experiences where the user can look in all directions. Another popular style of unstructured capture is to use internet photo collections. This provides an abundance of footage, certainly sufficient for room-scale VR experiences, but comes with several challenges: The source cameras are not known for many photos, making sparse reconstruction challenging [37, 38]. Furthermore, the photos have often been taken at different points in time, making 3D reconstruction problematic as the illumination can differ significantly and objects may have moved in the scene [39]. Finally, another interesting capture strategy is *unstructured video capture*, where a user records video while moving the camera to capture multiple viewpoints [40, 41, 42].

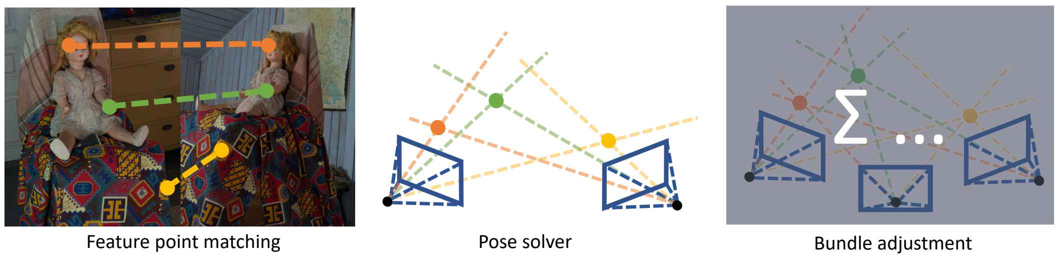


**Figure 2.7:** Common stages of a sparse reconstruction pipeline.

In this thesis, we employ unstructured capture strategies that can easily be performed by just one camera operator, and also provide footage suitable for VR experiences with 360° coverage of the scene. For seated VR experiences (Chapters 3-4), we use panorama-style capture where the user captures a burst of photos while rotating in place. For room-scale VR experiences (Chapters 5-6), we ingest an unstructured cloud of images — with the expectation that the scene has a large overlap and that each photograph has neighboring views with a sufficiently wide baseline.

### 2.1.4 Sparse reconstruction

After capture, it is crucial to obtain the 3D poses of the input images. We call this *sparse reconstruction* as it often also obtains a sparse 3D reconstruction of the scene. Figure 2.7 illustrates the three stages common to most sparse reconstruction methods: *Feature point matching*, *pose solvers*, and *bundle adjustment*. Here, we give a brief overview of each stage.

#### 2.1.4.1 Feature point matching

The primary signal which provides information about 3D camera pose and scene structure is 2D image motion. That is, we can infer how far away objects in the photos are and how the camera has moved by analysing the motion of pixels between two photos. Image motion, however, is a very difficult to measure, as simply finding nearby pixels with similar color values is often not good enough. A common technique is to, instead of attempting to track the motion of all pixels, find a number of salient 2D *keypoints*, which are easy to identify and match across multiple frames, and use only these points for 3D pose and structure.

This breaks down into several steps: Detection, description, and matching. First, *corner detectors* search the images for visually distinct elements, such as Harris corners [43] or Shi-Tomasi corners [44]. Then, *feature descriptors* are computed for each detected corner, summarizing the image content around the corner in as much detail as possible. To support a large variety camera motions, most commonly used descriptors, e. g., SIFT [45] and

SURF [46], are built to be invariant to rotation, scale, and changes in intensity. Finally, the *matching* step compares the descriptors between a pair of images to establish a set of corresponding keypoints. During this step, it is often crucial to reduce spurious correspondences using a several tests and checks, e. g., Lowe thresholding [45] which discards correspondences that either have very dissimilar descriptors or are likely to be ambiguous.

#### 2.1.4.2 Pose solver

Now that we have a set of 2D point correspondences across a pair of photos, we want to estimate the relative pose of the cameras. We can do so by using any number of algorithms designed specifically for this purpose, commonly known as pose solvers, which solve for relative extrinsics given a set of 2D point correspondences. Once these poses have been estimated, we can *triangulate* a 3D point for each matching pair of points.

Pose solvers need to be robust to incorrect correspondences, as feature point matching is often unreliable. Commonly, these approaches are based on the RANSAC algorithm [47], which chooses a random subset of the correspondences to compute the pose and uses the remaining correspondences to check the solution. After several iterations, the algorithm returns the most reliable pose. To make this more efficient, pose solvers have been developed that work with a minimal number of feature point correspondences as input: five correspondences if the camera intrinsics are known in advance [48], and seven correspondences for the general case [27, Chapter 11.3]. However, even with powerful outlier rejection techniques, pose solvers are unreliable for very small baselines [49]. In Chapter 4, we show that this can be mitigated if the depth of the feature point correspondences are known in advance.

#### 2.1.4.3 Bundle adjustment

While the poses and 3D points may be accurate for a particular pair of cameras, feature point matches often extend over a sequence of several images. This means that a single camera will often have many separate (and inconsistent) estimates for the 3D position of a point, and even sometimes inconsistent estimates for its pose. To coalesce these measurements, systems often perform *bundle adjustment* [50], an optimization strategy which attempt to minimize the distance between the projected 3D point across a set of images, by varying the 3D location of the point, as well as the 3D pose and intrinsic parameters of the input cameras. It is also common to account for lens distortion parameters during optimization.

Formally, bundle adjustment can be summarized as minimizing the *projection error* between the projections of 3D points and the 2D locations of their corresponding feature points in each camera. In other words, this amounts to solving

$$\underset{\{\mathbf{p}_i, \mathbf{M}_j, \mathbf{K}_j\}}{\operatorname{argmin}} \sum_i \sum_{j \in \mathscr{I}(i)} \rho(\mathbf{p}_{img}^{i,j} - n(\mathbf{K}_j \mathbf{M}_j \mathbf{p}_i)), \tag{2.8}$$

where $\rho$ is a robust cost function, $\mathscr{I}(i)$ is the set of images where the feature point $i$ has been detected, and $\mathbf{p}_{img}^{i,j}$ is the 2D location of $i$ in image $j$. In our case, the function $n$ only performs the perspective divide (see Equation 2.4), but it is often useful for this function to explicitly model geometric distortions from the camera lens.

#### 2.1.4.4 Structure from motion

*Structure from motion* (SfM) refers to the general class of methods which, provided with a set of images, will attempt to perform a sparse reconstruction using the components introduced in this section. These systems, while operating on arbitrary image collections, are designed to work for the general use-case, and therefore do not make any assumptions about the calibration, overlap, or locality of the input images. PhotoTourism [37] was the first SfM system to work on large collections of unstructured photographs, making it possible calibrate and localize collections of internet images, recovering the 3D structure of famous tourist landmarks. This system has later been extended to support larger data sets using either parallelisation over a cluster [38] or by reducing the time complexity of its components [51, 52]. Other extensions improve accuracy, e. g., with adaptive thresholds for robustly triangulating point correspondences [53]. Modern SfM systems are typically quite robust, and are the de-facto standard for sparse reconstruction in offline settings.

#### 2.1.4.5 SLAM

SLAM, or Simultanous Localization and Mapping, is a name for a class of methods originating in the robotics and autonomous mapping community. These methods are very similar to SfM, and the terms are often used interchangeably, but were explored concurrently by different fields of research, and thus are often specialized for different applications. More specifically, SLAM systems are typically designed for real-time online localization, meaning that they usually operate on video sequences instead of unordered image collections. While modern SLAM pipelines [54] closely resemble SfM systems, they often cannot afford to spend the same computational budget on bundle adjustment, because of the time constraints for online systems. This tends to produce reconstructions that are locally plausible, but less globally accurate when compared to SfM approaches.

## 2.2 Dense reconstruction

While the sparse reconstruction gives us 3D poses for each photo and a set of sparse points, it is common for viewpoint-free photography systems to also obtain a *dense* reconstruction of the scene, i. e., estimate depth at every single pixel. It is worth noting that there are viewpoint-free photography systems which do not rely on dense reconstruction. Indeed, Schum et al. [26] review several methods for viewpoint-free photography, placing them on a spectrum ranging from no use of dense 3D geometry [19] to approaches that rely on an explicit 3D reconstruction [55]. As the methods presented in this thesis use explicit 3D geometry, we review different approaches to dense 3D reconstruction.

### 2.2.1 Stereo matching

Here we describe a common approach to compute a dense reconstruction from photos: *Stereo matching* methods reconstruct 3D geometry by finding dense correspondences between the photos, i. e., they establish depth for every pixel in a photo by searching through neighboring photos for pixels with similar appearance.

#### 2.2.1.1 Photoconsistency

The core of most dense reconstruction methods is a way to estimate whether any given 3D point in the scene is likely to free space, or occupied by geometry. Stereo matching methods estimate this by projecting the 3D point into a collection of photos, and then checking if its appearance is similar — or *photoconsistent* — in each photo.

Most photoconsistency costs compare color similarity in local image patches. For example, to compute photoconsistency for a pixel $\mathbf{p}$ in an image $A$ at a certain depth, we reproject a small *fronto-parallel* patch $\mathscr{P}(\mathbf{p})$ around $\mathbf{p}$ (i. e., where every pixel shares the same depth) into adjacent photos $\mathscr{A}(A)$. We can then measure similarity using e. g., sum-of-absolute-differences (SAD):

$$\sum_{B\in\mathscr{A}(A)}\sum_{\mathbf{p}'\in\mathscr{P}(\mathbf{p})} |c_A(\mathbf{p}') - c_B(P_{A\to B}(\mathbf{p}'))|. \tag{2.9}$$

Here, the functions $c_A$ and $c_B$ fetch colors from the images $A$ and $B$.

There has been a large body of research on photoconsistency measures. Please refer to the tutorial by Furukawa and Hernandez for a comprehensive overview [25, Chapter 2].

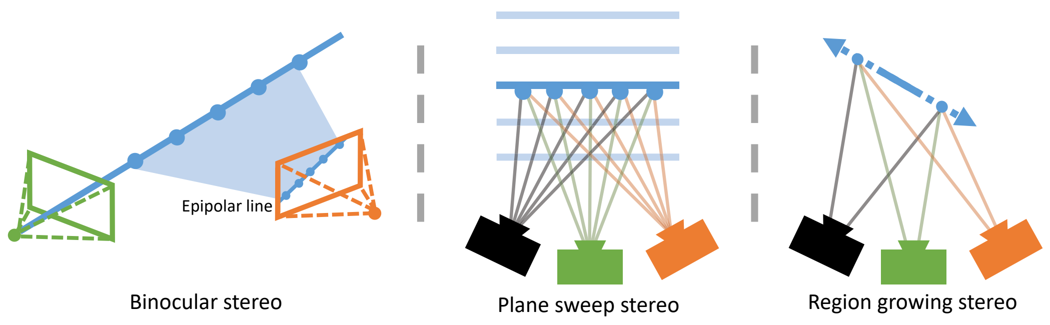


**Figure 2.8:** Illustration of different stereo matching strategies.

### 2.2.1.2 Binocular stereo

Binocular stereo estimates depth from a pair of photos, most often captured with a two-camera rig. This has several advantages. The camera poses and radiometric properties can be carefully calibrated in advance, resulting in higher accuracy. Further, the cameras can capture photos simultaneously, making it possible to reconstruct objects in motion. Finally, a lot of computation is saved, as photoconsistency only needs to be computed in two images.

As relative poses between both cameras are already known, a pixel will project onto a line in the other photo. This is called the *epipolar line*, see Figure 2.8 (left). Binocular stereo methods estimate depth by searching for along the epipolar line to for a location with a low photoconsistency cost. This is a well researched problem, with a large body of work focusing on both fast [56, 57] and accurate algorithms [58, 59].

### 2.2.1.3 Plane sweep stereo

We can extend the binocular stereo approach described above to work with multiple photos, resulting in the well known *plane sweep* stereo approach. Given a collection of camera poses and their respective photos, a plane sweep computes photoconsistency at a regularly spaced depths for each pixel in a reference photo. As Figure 2.8 (middle) shows, this is equivalent to computing photoconsistency on a collection planes that are fronto-parallel to the reference photo. This can efficiently implemented on modern graphics hardware [60]. Compared to binocular stereo, this multi-view plane sweep consumes more computational resources, but is less susceptible to outliers as photoconsistency is computed against more photos.

### 2.2.1.4 Region growing stereo

Instead of exhaustively computing photoconsistency for every pixel at every possible depth, it is possible to significantly speed up multi-view stereo matching by carefully selecting which

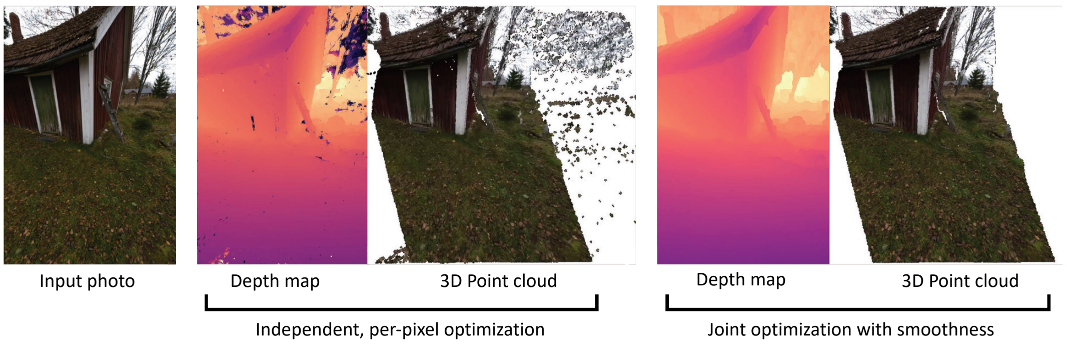

Input photo
Depth map
3D Point cloud
Independent, per-pixel optimization
Depth map
3D Point cloud
Joint optimization with smoothness

**Figure 2.9:** Enforcing smoothness during dense reconstruction often improves geometry in regions where photoconsistency is unreliable.

depths to try for a given pixel. *Region-growing* multi-view stereo approaches achieve this by searching for planes (i. e., both normals and depths) for each pixel during reconstruction, and establish new correspondences by iteratively growing the current set of planes to neighboring pixels, see Figure 2.8 (right). Both Goesele et al. [39] and Furukawa and Ponce [16] use robust feature point matching to find an initial set of sparse correspondences and employ non-linear optimization to refine depths and normals between iterations. PatchMatch Stereo [56] is a binocular stereo method that uses random search to both find new correspondences and refine existing ones, relying on region growing to propagate a good surface estimate throughout the images. Later efforts have extended this random search algorithm to the multi-view setting [61, 17].

### 2.2.2 Image processing techniques

It is worth noting that 3D reconstruction from photoconsistency has some well known limitations. For example, photoconsistency is ambiguous for textureless surfaces such as white walls, and cannot be trusted for surfaces with view-dependent appearance, e. g., for highlights on glossy surfaces. Even for diffuse surfaces with texture, photoconsistency may still struggle because of occluding objects in the scene and exposure differences between photos. As a result, the most photoconsistent depth may not be correct or consistent among pixels in a local region. To mitigate this, instead of independently selecting the best depth for each pixel, many approaches jointly consider the depth of all pixels in the image, making it possible to enforce smoothness in the resulting geometry. Effectively, this encourages the difference in depth between neighboring pixels to be small and often produces better geometry, see Figure 2.9.

There is a wide variety of techniques that enforce smoothness. Here, we only cover methods that work in image-space and make use of color content to preserve the structure of the scene, preventing foreground and background regions from merging. The techniques presented here are general purpose, and can be used beyond dense reconstruction, e. g., for image segmentation [62], optical flow estimation [63] or panorama stitching [64].

As reconstruction uncertainty grows larger further away from the camera (see Figure 2.5), the level of smoothing should be more aggressive for pixels with large depth values. Here we achieve this using an inverse depth parametrization, where we enforce smoothness on pixel *disparities* (see Section 2.1.1.3) instead of depths.

#### 2.2.2.1 Edge-aware filters

*Edge-aware filters* enforce smoothness by blurring the image content in an edge-preserving fashion. They compute a weighted average of the disparities in a neighborhood around each pixel $i$. In other words,

$$d(i) = \frac{1}{\sum_{j\in\mathcal{N}(i)} w(i,j)} \sum_{j\in\mathcal{N}(i)} w(i,j)d'(j), \tag{2.10}$$

where $d'(j)$ is an initial disparity estimate and $\mathcal{N}(i)$ is a neighborhood around the pixel $i$.

The weight function $w(i,j)$ depends on the specific type of edge-aware filter. For example, the commonly used cross bilateral filter [65] uses a product of two Gaussian kernels, one for image locations and one for colors:

$$w(i,j) = \exp\left(\frac{-||i-j||^2}{2\sigma_s^2}\right)\exp\left(\frac{-||c(i)-c(j)||^2}{2\sigma_c^2}\right) \tag{2.11}$$

where the function $c$ fetches colors from the image, $\sigma_s$ determines the spatial extent of the filter, and $\sigma_c$ determines how sensitive the filter is to image edges. Modern edge-aware filters run much faster, e. g., the *guided filter* [66] whose run-time does not depend on the spatial extent of the filter kernel.

**Weighted median.** For dense reconstruction, it is often detrimental to use edge-aware filters that compute weighted averages, as this runs the risk of connecting the foreground and background with spurious average disparity values near object boundaries. Instead of computing an average, *bilateral median filters* [67] alleviate this issue by using the weights to pick a representative, weighted median disparity from the neighborhood around each pixel.

#### 2.2.2.2 Markov random fields

*Markov random fields* (MRFs) enforce smoothness for discrete labelling problems. This is particularly useful for plane-sweep stereo approaches, where we are tasked with picking a disparity for each pixel from a discrete set of planes. Formally, this is expressed as finding a disparity label $d(i)$ for each pixel $i$ that minimizes the energy

$$\sum_i E_p(i, d(i)) + \sum_{i,j \in \mathcal{N}} \lambda_s(i,j) E_s(d(i), d(j)), \tag{2.12}$$

where

- $\mathcal{N}$ is the set of all pairs of neighboring pixels $(i, j)$,
- $E_p(i, d(i))$ is a photoconsistency cost for $i$ at the disparity $d(i)$,
- $\lambda_s(i, j)$ is a balancing weight for the smoothness energy, and
- the smoothness energy $E_s(d(i), d(j))$ prefers nearby pixels to have similar disparity.

Common forms for the smoothness energy are, e. g., the Potts energy

$$E_s(d(i), d(j)) = 1 \text{ if } d(i) = d(j) \text{ else } 0, \tag{2.13}$$

which only encourages exact label matches, or a truncated linear energy

$$E_s(d(i), d(j)) = \min(|d(i) - d(j)|, k), \tag{2.14}$$

for some $k > 0$, which allows for the label to gradually change across the image.

Instead of using a global weight $\lambda_s(i, j) = \lambda_s$, it is useful to make the smoothness energy *edge-aware*, which encourages label changes at strong image gradients, e. g.,

$$\lambda_s(i, j) = \exp\left(\frac{-||c(i) - c(j)||^2}{2\sigma^2}\right), \tag{2.15}$$

where $c$ fetches colors from the photo, and $\sigma$ determines how sensitive the smoothness cost should be to image gradients.

**Optimization.** There is a vast number of algorithms to minimize energy functions of this form. The alpha-expansion algorithm [68] is a common choice. If performance is a concern, fast and approximate algorithms such as semi-global matching have been shown to work well for dense reconstruction [69].

#### 2.2.2.3 Linear systems

Smoothness can also be enforced using a *linear system*. Compared to the MRF formulation above, linear systems can solve for continuous disparity values, but are restricted to energies that can be expressed as a sum of squared distances. Formally, these systems solve for disparity $d(i)$ for each pixel $i$ to minimize the sum

$$\sum_i ||d'(i) - d(i)||^2 + \sum_{i,j \in \mathcal{N}} \lambda_s(i,j) ||d(i) - d(j)||^2, \tag{2.16}$$

where $d'(i)$ is an initial disparity assignment for each pixel. Similarly to the MRF formulation above, it is useful to make the smoothness energy edge-aware, to encourage stronger smoothness in areas without texture.

**Optimization.** Linear systems can be very efficiently solved using modern optimizers based on the conjugate gradient method [70], which can be implemented on graphics hardware [22].

#### 2.2.2.4 Convolutional neural networks

While not explicitly enforcing smoothness, many recent techniques formulate dense reconstruction as a machine learning problem, to be solved using *convolutional neural networks* (CNNs) [71, 59]. Here, we only give a brief functional summary of CNNs. For a more thorough treatment, please refer to the course notes by Karpathy et al. [72].

A CNN is a function composed of several consecutive convolution operators, where the weights in each convolution kernel together form a set $\Theta$ of learnable parameters. Since CNNs consist only of convolutions, we can model it as a function which forms the output image $O$ by processing each pixel $i$ independently, using only a patch $\mathscr{P}(i)$ around $i$ as input:

$$O(i) = f(\mathscr{P}(i), \Theta). \tag{2.17}$$

The size of the input patches $\mathscr{P}(i)$ is called the *receptive field*, and depends on the architecture of the network. A large receptive field is often desirable, as it enables the network output to depend on context information around each pixel.

The parameters $\Theta$ are found by minimizing a loss function $\mathscr{L}$ on a training data set $\mathscr{T}$ with known input $I$ and output $O$ images, i. e.,

$$\Theta = \operatorname*{argmin}_{\Theta} \sum_{(I,O) \in \mathscr{T}} \mathscr{L}(f(I,\Theta), O). \tag{2.18}$$

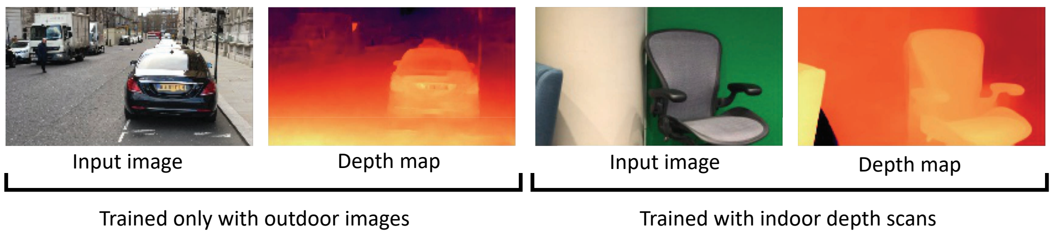

Input image
Depth map
Input image
Depth map
Trained only with outdoor images
Trained with indoor depth scans

**Figure 2.10:** Example depth maps from a single-image depth estimation network. **Left:** The Monodepth method [81], trained without ground truth depth maps, using only photo-consistency between images as training signal. **Right:** The same network, but trained with a standard loss against indoor depth scans in the ScanNet dataset [76].

Not unexpectedly, the exact formulation of the loss function $\mathscr{L}$ depends on the application.

CNNs are incredibly powerful tools, and have been successfully used in state-of-the-art approaches for binocular stereo [59] as well as other applications, such as image classification [73] and optical flow [74]. However, as $\Theta$ often consists of millions of parameters, these approaches generally use large training data sets containing several thousand input-output image pairs [75, 76]. For dense reconstruction, such datasets are particularly problematic to build, as obtaining ground-truth geometry for an image is a time-consuming and challenging task [77].

### 2.2.3 Depth without correspondence

There are several methods of acquiring depth estimates for a photo without explicitly estimating correspondence between pairs or sets of photos. These methods typically require other forms of complementary information, like projected light patterns. In recent years, many newer methods are able to estimate depth from single images without any additional hardware, through the use of deep learning.

#### 2.2.3.1 Depth sensors

Depth sensors actively measure the depth by projecting light onto the scene. One method to acquire depth is with a structured light sensor such as the Microsoft Kinect for Xbox 360 [78, 79]. These sensors project a light pattern onto the scene and recover depth by measuring how the pattern deforms in an image taken by a camera on the sensor. Another solution is with a time-of flight sensor, such as the newer Microsoft Kinect for Xbox One [80]. One significant drawback with depth sensors is that they have limited range and seldom work outdoors in bright sunlight, as they rely on projecting light onto the scene.

#### 2.2.3.2 Single-image depth

Depth estimation from a single image is an ill-posed problem: it is impossible distinguish a image of a real 3D environment from a image of a photograph. Yet, a human observer can easily understand the 3D structure of a scene from just a single image. Recent efforts have successfully used CNNs to emulate this behaviour, and estimate dense depth using just a single image as input. See Figure 2.10 for examples. While these approaches show a lot of promise, they generally work best in the type of environments that are well represented in the training data. Early approaches used depth sensors to capture training data [82], resulting in networks that specialize on indoor scenes where depth sensors work best. Li and Snavely [83] present a method which works well on a larger variety of scenes, by using MVS reconstructions of famous tourist landmarks as training data. It is also possible to completely circumvent the need for geometry as training data, by training the CNN to perform view synthesis, where depth is used as an intermediate output [84, 81]. During training, the estimated depth is used to warp the input photo to match the appearance of other photos (i. e., minimize a photoconsistency loss), taken at different viewpoints in the scene.

In Chapter 4, we show that single-image depth is feasible for viewpoint-free photos.

### 2.2.4 Geometry fusion

Using the methods above we can compute dense depth for every pixel in a photo, forming a so called *depth map*. However, independently computing depth maps for each photo in a scene is often undesirable. It results in redundant computation and storage, as the depth for a given location in the scene is estimated several times — once per photo. Furthermore, it is possible for the resulting depth maps to disagree, especially in regions where photoconsistency is unreliable, e. g., textureless surfaces or highlights. Here we discuss methods to alleviate these issues, by exploiting redundancy to discard incorrect depth estimates, and eventually eliminating redundancy by fusing the depth maps into a global representation.

#### 2.2.4.1 Geometric consistency

Geometric consistency filters exploit redundancy between the depth maps to detect and discard spurious depth values, see Figure 2.11. This is a common operation used in several dense reconstruction systems [85, 86, 16, 87, 17].

These filters rely on the fact that depth maps computed with multi-view stereo tells us much more than just the 3D location of every pixel: 1) The volume between the camera location and the surface in the depth map is *free space*, and 2) for MVS reconstruction to

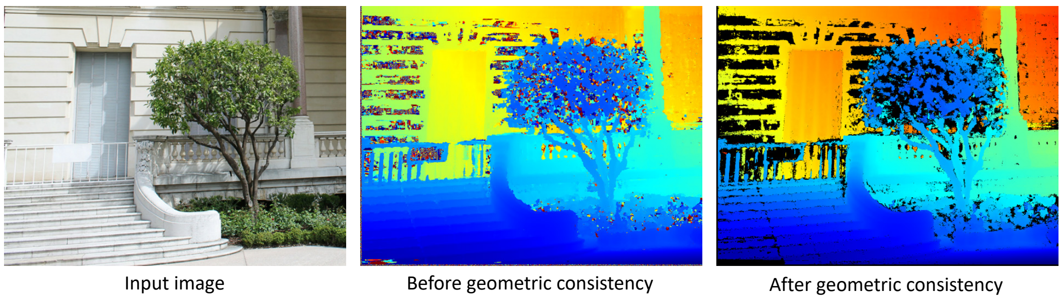


**Figure 2.11:** The effect of the geometric consistency filter in COLMAP [17]. Note how it discards spurious depth values (shown as colored speckle noise).

be feasible, a pixel needs to be visible in a least two photos. Thus, at least two depth maps should agree on the 3D location of a pixel. Based on this, these filters enforce geometric consistency by discarding pixels whose estimated 3D location:

1. is in a region considered to be free space by several other images, and
2. is not corroborated by any other depth map.

In this thesis, we enforce geometric consistency by reprojecting each pixel into several neighboring photos, where we use a threshold $\tau_r > 1$ on the ratio

$$r = \frac{z_r}{z_n} \tag{2.19}$$

between the depth of the reprojected pixel $z_r$ and depth map value $z_n$ in the neighboring photo to determine whether this is free space conflict ($\tau_r > r$, i. e., $z_n > z_r$), or whether the two depth maps agree ($\tau_r^{-1} < r < \tau_r$, i. e., $z_r \approx z_n$). We then discard pixels that either have too many free space conflicts ($> \tau_f$) or have been corroborated by too few neighboring depth maps ($< \tau_c$). As an optional clean-up step, we smooth out salt-and-pepper noise with a weighted median filter [67], guided by disparity.

### 2.2.4.2 Building a global representation

Many dense reconstruction systems contain a step which combines all depth maps into a single, global representation for geometry [88, 89, 90]. This is useful, as it discards many outliers, eliminates redundancy, and often outputs a triangular mesh which can be rendered using standard graphics engines.

While it is feasible to convert all depth maps into triangle meshes and stitch these together in 3D [91], this is challenging as it requires us to explicitly resolve connectivity

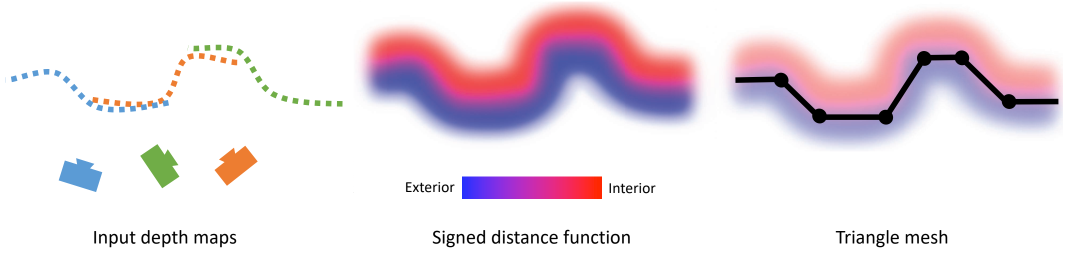


**Figure 2.12:** Overview of geometry fusion using signed distance functions.

between points. Moreover, these methods are often sensitive to noise in the input data [85]. Instead, a common approach is to build a *volumetric* reconstruction of the scene: As illustrated in Figure 2.12, for each 3D point in the scene, we determine whether the space surrounding it is either empty (i. e., *exterior*) or occupied (i. e., *interior*). We can then extract the combined geometry as the boundary between empty space and occupied space, using e. g., the Marching Cubes algorithm [92].

Here, we briefly introduce common volumetric reconstruction approaches based on two design decisions: 1) how to *represent* interior and exterior for each 3D point in the scene, and 2) how to *construct* this representation from a collection of depth maps?

**Representation.** One possible representation is a hard binary segmentation the scene into interior and exterior regions. While straightforward, this representation has been successfully used by many recent volumetric fusion approaches [93, 94]. Alternatively, we can use a representation that allows for uncertainty, where we instead express how likely each point is to be either interior or exterior. One popular such representation is the *signed distance function* (SDF) [85]. That is, at each point in the scene, we store the *distance* to the closest surface, using negative values to represent exterior (*signed*).

An important consideration is how to quantize 3D space, so we can digitally represent 3D functions, e. g., the binary segmentation and SDF described here. One option is to *voxelize* the scene, i. e., represent it as a regular grid of cubes. Instead of cubes, other primitives can be used. Tetrahedra are a common choice, as they make it possible to irregularly quantize the scene. This is often achieved using Delauney tetrahedralization, which adapts the size of tetrahedra to the local density of 3D points from the depth maps [93, 94].

**Construction.** The central challenge for volumetric reconstruction approaches is how to build one of the above representations from a collection of imperfect depth maps.

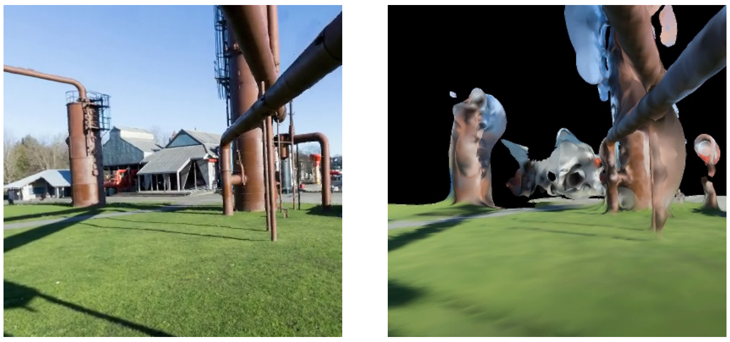


**Figure 2.13:** Issues with fused geometry used directly used as viewpoint-free photographs. **Left:** Reference photo. **Right:** Same view, showing geometry built with MVS depth maps from COLMAP [17] fused with Screened Poisson Surface Reconstruction [96]. Note the blobby appearance near image edges and that far-away regions are missing.

The SDF representation can be rapidly built using straightforward averaging [85, 88]: First, the SDF of each depth map is approximated by only searching for the nearest surface along the line of sight for every pixel. Then, the global representation is computed as the pointwise average of the resulting SDFs. A drawback with this approach is that it creates holes in regions with missing data or where depth maps disagree [86]. This can be addressed by solving for the distance function with a 3D linear system, where surface normals can be incorporated to extrapolate surfaces and plug holes [95, 96]. However, this is sensitive to unreliable surface normals, a problem which can partially be addressed by incorporating free space constraints in the optimization [97].

The binary segmentation representation is popular for MVS systems [93, 98, 99, 94, 90]. The idea is to solve for this segmentation using a 3D MRF, where camera locations are forced to be exterior regions, and the space behind every depth map pixel is encouraged to be interior. While some approaches use a regular voxel grid [98], most publicly available 3D reconstruction systems compute this segmentation on a Delauney tetrahedralization of the scene [94, 90, 100]. In this thesis, we make heavy use of the RealityCapture system [18], which is a commercial evolution of one of these methods [94].

**Discussion.** These geometry fusion approaches often yields unsatisfactory results if the resulting geometry is directly used as viewpoint-free photographs. As Figure 2.13 shows:

1. These approaches have to compromise between depth maps, which often results in blobby geometry that does not align well with edges in the input photos.

2. These fusion approaches tend discard noisy depth estimates, where the triangulation angle is narrow. As a result, many algorithms fail to recover far-away geometry.

## 2.3 Representations for rendering

In the sections above, we described methods that reconstruct the geometry of a scene. However, the focus of this thesis is on capturing viewpoint-free photographs, i. e., photos where we can realistically change the viewpoint after capture. To achieve this, we need a representation for the visual information in the scene, which often has to describe more than geometry, due to effects such as: view-dependent materials (e. g., highlights), reflections, refractions, and light scattering through participating media (e. g., light cones in fog).

Unsurprisingly, the choice of representation strongly affects many of our goals from Chapter 1. It imposes an upper bound on realism (goal 3), e. g., if the representation does not support reflections, we cannot realistically capture a scene with mirrors. This choice also has practical implications for rendering performance (goal 4) and range of motion (goal 2). This also imposes requirements on the earlier stages of viewpoint-free photography pipelines. For example, a representation which relies on many input photos needs a convenient capture strategy and a scalable approach to sparse reconstruction [29]. Alternatively, representations that require high quality geometry rely heavily on the dense reconstruction stage [101].

In this section, we review several representations for viewpoint-free photos, and evaluate how suitable they are in the context of our goals.

### 2.3.1 Artifacts

It is difficult to compare viewpoint-free photography representations in terms of quality and realism (goal 3). Indeed, it is a challenging research problem to automatically detect distortions in photos [102], even if we have access to clean reference photographs [103]. To concretely discuss both quality and realism, we compare representations in terms of how effectively they can avoid common visible mistakes, or so-called *artifacts*.

In Figure 2.14, we illustrate many frequent types of artifacts:

**Floating geometry:** Some stereo matching approaches incorrectly reconstruct foreground depth for regions where photoconsistency is unreliable (e. g., with low overlap, strong highlights or scene motion), resulting in ”floating” pieces of geometry.

**Misaligned occlusion edges:** Many geometry fusion approaches either inflate or erode foreground objects in the scene. This causes misalignment between the scene geometry and the input photographs at *occlusion edges*, i. e., the boundary between foreground and background. This is sometimes referred to as *edge-fattening* in stereo literature.

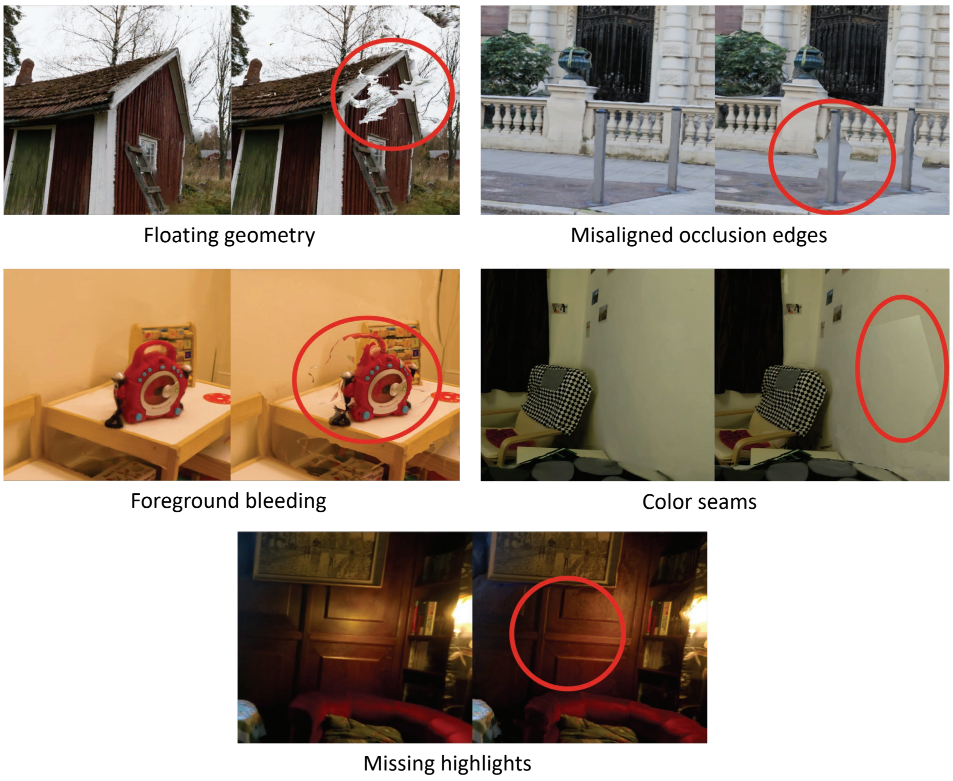


**Figure 2.14:** Example artifacts commonly faced for viewpoint-free photography methods.

**Foreground bleeding:** Inaccurate foreground geometry can also lead to severe ghosting artifacts, where the color of foreground objects is projected onto background geometry.

**Color seams:** These seam artifacts are often caused by misalignment, different resolutions, and shifts in colors for the same (or neighboring) content across input photos.

**Missing highlights:** Highlights are important for realism. However, it may often be more visually pleasing to discard highlights that cannot be convincingly be reproduced. This is particularly important for methods that target easy capture, and thus work with a limited amount of input data.

Finally, as viewpoint-free photography is intended for interactive exploration, the output imagery needs to be temporally stable.

### 2.3.2 Panoramas

Panoramas are 2D photos with a wide field of view. Using dedicated consumer hardware, such as the Ricoh Theta, it is now easy to capture full 360° $\times$ 180° panoramas. Although this representation is often marketed for VR experiences, panoramas provide no range of motion as they only capture a single viewpoint in the scene (goal 2).

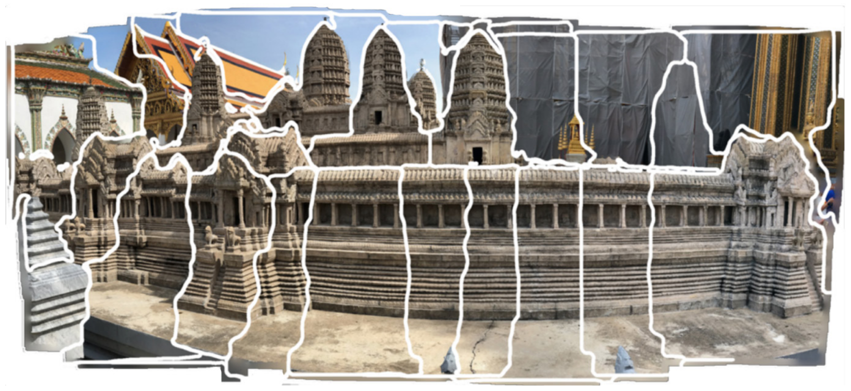


**Figure 2.15:** Panorama stitched from multiple photos using seam-hiding techiques. Note how the seams (white) align with color edges where they are difficult to notice.

#### 2.3.2.1 Parallax-tolerant panorama stitching

Panorama stitching methods build panoramas by combining multiple input photos. Many such methods align input photos assuming rotation-only motion, which often results in misalignment artifacts such as seams or ghosting. These artifacts can be alleviated with *seam-hiding* stitching techniques, see Figure 2.15. For example, by solving for the output panorama with an MRF, where the smoothness energy encourages seams between input photos to occur where they are hard to detect, i. e., where the photos are visually similar [104, 105].

Later methods compute warp-deformations to compensate for parallax-induced misalignment between the input images. Zhang and Liu [106] stitch image pairs with large parallax by finding a locally consistent alignment sufficient for finding a good seam. Perazzi et al. [107] extend this work to the multi-image case and compute optimal deformations in overlapping regions to compensate parallax and extrapolate the deformation field smoothly in the remaining regions. Lin et al. [108] handle two independently moving cameras whose relative poses change over time.

These approaches produce better looking panoramas with fewer artifacts. However, they cannot be directly used for viewpoint-free photography, as they do not address the fundamental limitation that panoramas do not support viewpoint changes at runtime (goal 2).

#### 2.3.2.2 Stereo panoramas

Omnidirectional stereo (ODS) encompasses a class of techniques for stitching together a *stereo panorama* — a pair of left-eye and right-eye panoramic images [109, 9, 10, 30]. The original techniques for generating such images used strips from moving camera images to create the panoramic images, using either mechanical rotation [109, 110] or hand-held images

[10, 111, 12]. More recent systems enable the capture of stereo panoramic videos using multiple cameras arranged in a ring [30, 112], or using two spinning wide-angle cameras [113]. Unfortunately, stereo panoramas have a number of drawbacks [30], particularly when applied to full spherical images. These include:

1. non-linear perspective: straight lines in the scene are rendered as bent;
2. unnatural non-rigid motion: the scene appears to swim or bend above and below the equator as the viewer turns their head;
3. the lack of realistic motion parallax as the head is turned;
4. incorrect stereo parallax away from the equator, most noticeable as the distance to the floor appearing “at infinity”, resulting in unpleasant vertigo;
5. the inability to tilt the head sideways and obtain correct binocular parallax;
6. un-specified behavior at the poles: a simple 180 vertical sweep leaves gaps.

There are some other stitching methods that are not specifically designed for ODS but general stereoscopic image pairs. Luo et al. [114] describe a method for cloning a *manually* specified selection from one stereoscopic image to another, and adjusting the colors and disparities through iterative gradient-domain blending.

#### 2.3.2.3 Depth panoramas

An alternative to generating a left-right pair of panoramic images is to augment a traditional stitched panoramic image with depth information [115]. While this somewhat increases the range of motion by enabling viewpoint changes during run-time, adding a single depth channel to a panorama is often not sufficient, as it often exposes artifacts at depth discontinuities, i. e., reveal long stretched triangles or holes.

Zheng et al. [116] create a *layered depth panorama* from multiple overlapping images using a cylinder-sweep MVS algorithm. One problem with Zheng et al’s approach is that their layers are not connected, which prevents their algorithm from reproducing sloped surfaces. Thatte et al. [117] describe a representation rendering for ODS with depth and sketch out capturing and rendering algorithms, but mostly demonstrate their system with synthetic data, leaving the question of reliable depth estimation open.

### 2.3.3 Light fields

Like panoramas, light field representations for viewpoint-free photos make no use of scene geometry. However, while panoramas can only express the color of light rays that converge on a single point, light field representations allow for viewpoint changes, as they model all light rays that pass through a volume in space. This results in realistic, high-quality viewpoint-free photos with support for view-dependent effects. However, it requires a lot of storage. In the general case, this would require us to discretize and store a five dimensional function — essentially a regular 3D grid of 2D input photos. The Light Field [19] and the Lumigraph [20] representations reduce this to four dimensions by assuming that the viewing volume only contains empty space. This can be further reduced to three dimensions, if the output camera motion is confined to a planar disk [118].

The main disadvantages of these representations is that they have a limited range of motion (goal 2), and require a very large number of input photos, which makes the capture process tedious and time consuming (goal 1). This can be somewhat alleviated with hybrid techniques that establish correspondences between the input photos, making it possible to synthesize in-between views using image warping. This is particularly useful for viewpoint-free *video*, where image warping can be used to interpolate between a sparse set of input camera locations [34].

Early hybrid approaches used depth-based image warping to build light fields from hand-held capture. At first, purpose-built capture stages were needed to estimate the scene geometry [20], while later methods could instead rely improvements in sparse and dense reconstruction [119, 120]. CNN-based approaches have also been developed to generate dense light fields from a limited number of input photos [121, 122]. However, these approaches focus on object capture, resulting in small light fields with a limited field of view. Recently, image warping has made it possible to build larger, panoramic light fields from hand-held footage, although the output camera motion is restricted to a disk [42]. Even with geometry from modern 3D reconstruction techniques, motorized capture rigs are still necessary to build light fields with a large enough range of motion for seated VR experiences [29].

While it is feasible to build room-scale light fields using robot capture, the resulting representation is not directly suitable for VR experiences, as the output camera location is constrained to a plane [15].

### 2.3.4 Global texture

The representations above assume that we have no geometric information of the scene. However, it is often useful to incorporate such information into representations for viewpoint-free photos. A common way to achieve this is to apply a global *texture map* onto a dense

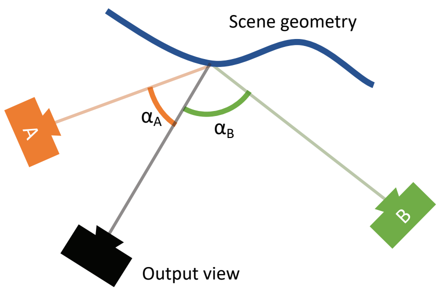


**Figure 2.16:** The geometry behind view-dependent blending. Output view shown with black, in a scene with two input photos, A and B.

reconstruction of the scene. That is, overlay a high resolution 2D image on top of the surfaces in the scene. This texture map is commonly obtained by projecting input photos onto the scene geometry and resolving alignment artifacts using e. g., seam-hiding stitching [123], warp deformations [124], or super-resolution techniques [125].

This is representation can easily be rendered from any viewpoint in the scene using standard graphics engines, resulting in real-time display rates (goal 4) and, in theory, a large range of motion (goal 2). However, quality heavily depends on the accuracy of the reconstructed geometry, which for MVS reconstructions is often insufficient, resulting in holes and occlusion edge artifacts. Furthermore, a global texture cannot represent view-dependent effects. Even with perfect geometry this approach looks artificial, as highlights are baked in or missing completely.

While most texture maps are commonly applied on top of triangle meshes, it is worth noting that they can also be used with other representations for scene geometry, such as point clouds [126]. Compared to triangle meshes, which require several post processing operations, point clouds have potential for more accurate occlusion edges, as they stay closer to the original MVS depth maps. However, point clouds are expensive to display, as they do not store connectivity between points, and instead need to resolve it on the fly.

### 2.3.5 View-dependent textures

Compared to the global texture representation above, applying *view-dependent textures* onto scene geometry has potential for higher realism. As the name implies, these representations apply a texture map which changes when the camera moves.

As illustrated by Figure 2.16, these approaches compute the texture map by blending together photos that have been projected onto the scene geometry. That is, each color $c$ in the view-dependent texture map is computed on the fly as a weighted average

$$c = \frac{1}{\sum_i w_i} \sum_i w_i c_i \tag{2.20}$$

over the input photos $i$, where the blend weight $w_i$ determines how suitable each input photo is for the output viewpoint. Commonly, the blend weights are defined as a function $w_i = f(\alpha_i)$ of the angle between the line of sight to the output view and the line of sight to the input photo.

Early approaches were designed for either a few input photos captured by hand [55], or an abundance of photos from a capture rig [101]. The Unstructured Lumigraph [127] algorithm works with any number of input photos, and generalizes into a light field representation if enough photos are used.

While these representations are able to reproduce view-dependent effects, they still rely on the accuracy of the reconstructed geometry. Some artifacts caused by inaccurate geometry, such as blurring and ghosting can be alleviated with soft visibility [128] and image-warping techniques [129]. However, artifacts are still visible at misaligned occlusion edges.

### 2.3.6 Per-input-view geometry

It is possible to reduce artifacts at occlusion boundaries by tailoring custom geometry for each input photo, as this makes it easier to align geometry with image content. This has lead to representations for viewpoint-free photos that use *per-input-view geometry* to warp input photos into the output view, forming the image as a view-dependent blend of the warped input photos.

This was first demonstrated in synthetic scenes with accurate geometry [130], with later approaches proving it feasible to build a similar representations from real photos using stereo matching [131]. Recent approaches improve quality for real-world scenes with representations that enforce smooth pixel motion when warping photos, except at occlusion boundaries. This is commonly achieved by splitting input photos into segments whose boundaries align with strong image edges, under the assumption that these also align well with occlusion edges. Then, smooth pixel motion is enforced within each segment by, e. g., solving for constant [36] or planar depth [21] within the segment. Alternatively, this can be enforced at a later stage, by computing as-rigid-as-possible pixel motions when warping the segments into the output image [23, 132].

Similarly to view-dependent textures, these representations support view dependent effects and highlights. However, this requires extensive capture, as highlights need to overlap in several input photos, so that cross-fading creates the appearance of highlights moving plausibly across surfaces in the scene. Per-view representations circumvent this as they can unwrap the reflection and compute larger depths for highlights and reflections, enabling smooth interpolation [22].

However, these representations are expensive to render, especially in large scenes with several input photos, as they use custom geometry for each photo. This also limits range of motion, and most approaches only demonstrate close to linear output camera motion in small scenes captured with only a handful of photos [23, 22, 132]. Furthermore, while per-view geometry can make use of image edges to better align geometry at occlusion edges, these representations also introduce more opportunities to make mistakes. This often results in artifacts like floating geometry, and does not completely eliminate foreground bleeding and misaligned occlusion edges.

Even so, this representation provides an interesting solution to gracefully degrade quality far away from input photos. Ambient Point Clouds [133] achieve this by applying a “non-photorealistic” look wherever the reconstructed geometry is unreliable.

### 2.3.7 View-dependent geometry

Instead of computing geometry for each *input* view, some representations instead tailor the geometry for every *output* view. This is more flexible than the above approaches, which rely on static geometry that has been computed in a pre-processing stage.

Building geometry at each output view allows it to be more optimal for the particular novel viewpoint. More specifically, cues such as image priors can be used to guide geometry reconstruction in a way that produces rendered images which are most plausible, given the known input images. This can be done either by encoding the priors as a dictionary of patches from input photos [134, 135, 136, 137], or by learning a prior from data, encoded as a CNN [71]. While these approaches produce high quality output images, they are far from interactive, as building new geometry for each output view is time consuming.

Other techniques compute per-input view information to augment the view-dependent representations. Instead of estimating explicit scene depth, direct image correspondences can be used to simulate 3D camera motion [138], and hybrid approaches show that its possible to decompose these correspondences into depth and scene motion [139], making viewpoint-free photography possible in dynamic scenes. Correspondence-only image interpolation has also proven to be feasible with CNNs [140], albeit with strong visual artifacts.

Other methods avoid explicit scene depth estimation by using *soft* representations. Instead of committing to a single depth value for each input pixel, the recent Soft3D algorithm [24] computes a *soft* 3D reconstruction of the scene, encoded as a plane sweep volume, and produces output images by ray marching through a version of this representation that has been resampled to match the output view. This produces high quality results when staying close to input photos, but only supports a limited range of motion before discretization artifacts occur.

Chapter 3

# Casual 3D Photography

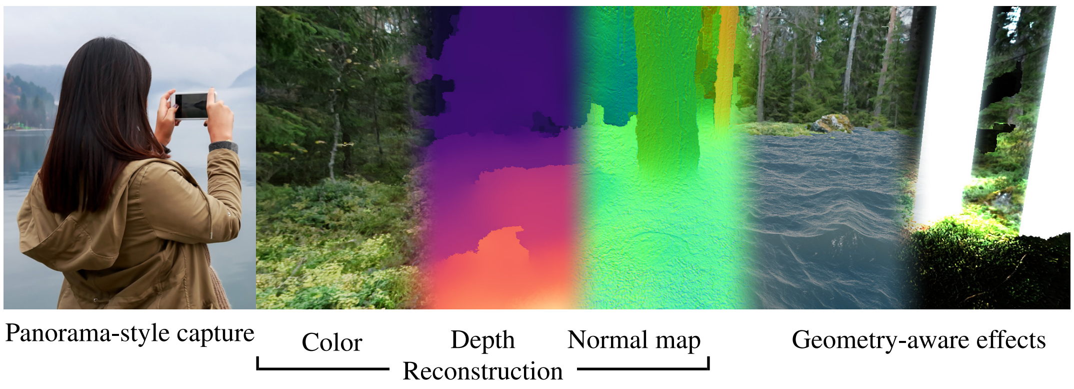


**Figure 3.1:** Using casually captured smart phone or DSLR images as input, our algorithm reconstructs a viewpoint-free photo suitable for seated VR experiences. The result is a *3D photo*, i. e., a multi-layered panoramic mesh with reconstructed surface color, depth, and normals. 3D photos can also be viewed on a regular mobile device or in a Web browser, and the reconstructed depth and normals allow interacting with the scene through geometry-aware effects.

In this chapter, we present a technology which creates viewpoint-free photos for seated VR experiences. In particular, we describe a suitable representation and introduce reconstruction algorithms specifically designed for easy capture.

We do not require the user to capture a scene from all angles, but just an arc around a single viewpoint. A person captures a scene by moving a hand-held camera sideways at about a half arm's length, while taking a series of still images (Figure 3.1, left), possibly with the help of a dedicated capture app. The capture is unstructured, i. e., the motion does *not* have to be precisely executed, and takes just seconds to a few minutes, depending on the desired amount of coverage. Given this input, our algorithm automatically reconstructs a *3D photo*, i. e., a textured, panoramic, multi-layered geometric mesh representation for viewpoint-free photos specifically designed for seated VR (Figure 3.1, middle).

Our 3D photos look best when viewed from within a volume around the viewpoints that were spanned during capture. The most immersive way to enjoy a 3D photo is in VR. Using the reconstructed geometry, we render stereoscopic perspective views that correctly react to tracked user head motion, i. e., provide binocular and motion-parallax depth cues. On non-VR displays such as smart phones, we can still show motion parallax by rendering from viewpoints on a sphere fitted to the estimated input camera locations. Our geometric representation enables interacting with the 3D photo through geometry-aware effects (Figure 3.1, right).

An alternative easy-to-capture representation often used for VR is *omnidirectional stereo* (ODS). This representation provides a binocular depth cue by delivering different images to the eyes, but is not suitable for seated VR experiences as it does not provide motion parallax when the user moves their head. In contrast, our representation provides binocular *and* motion parallax depth cues, and therefore a better sense of immersion. Refer to Section 2.3.2.2 for a more in-depth description of the limitations of the ODS representation.

The mature field multi-view stereo (MVS) has seen over 30 years of research, and several high quality and actively maintained software packages implementing state-of-the-art algorithms are available (see Section 2.2.1). However, applying these algorithms directly in our scenario produces unsatisfactory results, for the following reasons: (1) our casually captured images violate many common assumptions in MVS algorithms leading to geometric artifacts: they are captured with a narrow baseline, and our scenes are often not fully static and contain large textureless areas; (2) the geometry produced by MVS algorithms is not optimized for a specific viewpoint and often lacks completeness and detail. Our approach uses state-of-the-art reconstruction algorithms as core components, but through several technical innovations, we make them work robustly in our scenario. We provide extensive comparisons against several MVS packages in Section 3.3.2 and the supplementary material.

Our 3D photo representation can be rendered on any modern platform using standard graphics engines. The reconstruction quality is sufficient for moderate viewpoint changes, roughly within the volume spanned by the captured images. We have experimented with a variety of playful geometry-aware effects that make use of the reconstructed scene geometry.

We have applied our algorithm to numerous 3D photos captured with DSLRs and cell phone cameras. Among these are indoor and outdoor as well as man-made and natural scenes. We compare our 3D photos extensively with results obtained with existing state-of-the-art

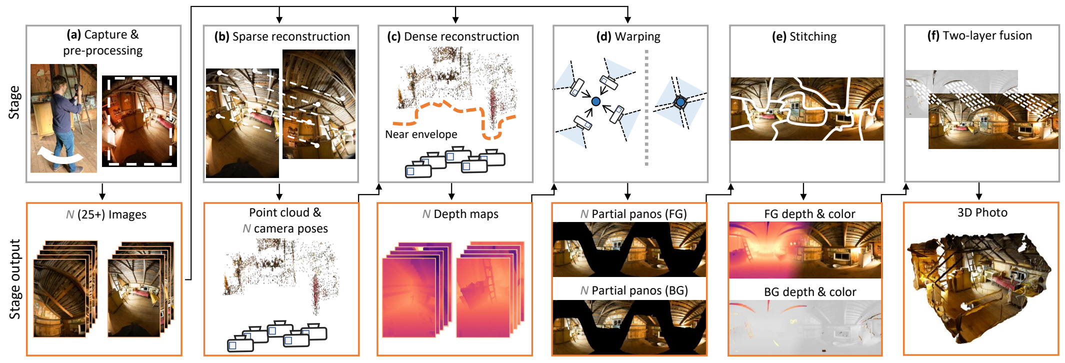


**Figure 3.2:** A breakdown of the 3D photo reconstruction algorithm into its six stages, with corresponding inputs and outputs: **(a)** Capture and pre-processing, Section 3.2.1; **(b)** Sparse reconstruction, Section 3.2.2; **(c)** Dense reconstruction, Section 3.2.3; **(d)** Warping into a central panorama, Section 3.2.4.1; **(e)** Parallax-tolerant stitching, Section 3.2.4.2; **(f)** Two-layer fusion, Section 3.2.4.3.

reconstruction algorithms, and provide quantitative analysis using the Virtual Rephotography evaluation method [141]. In the supplementary material, we provide datasets consisting of input images, results, and intermediate algorithm stage outputs for all of our scenes, as well as the results and rephotography error maps for 16 variants of competing methods.

## 3.1 Overview

One of our primary design goals is to make the 3D photo capture process easy for inexperienced users: the capture should be hand-held, should not take too long, and should be captured with an existing, low-cost camera. These requirements influenced many of the algorithmic design decisions further down the pipeline.

The input to our reconstruction algorithm is a set of captured photos. We do not require them to be taken in any particular way, as long as they contain sufficient parallax and overlap to be registered by a standard structure from motion algorithm. In practice, it works best to capture while moving the camera on a sphere of about half arm's length radius. While we captured the images manually to produce the results in this chapter, we envision a dedicated capture app could further simplify and speed up the process.

We represent a 3D photo in a single-viewpoint panoramic projection that is discretized into a pixel grid. While we use an equirectangular projection in our representation, other sensible choices, such as a cube map, would also be possible. Every pixel can hold up to two layers of "nodes" that store RGB radiance, normal vector, and depth values. This resembles layered depth images [142]. However, in addition to the layered nodes, we also store the connectivity between and among the two layers, in a manner similar to that of [143].

Our representation has several advantages: (1) the panoramic domain has an *excellent resolution trade-off* for rendering from a specific viewpoint, since it provides automatic level-of-detail where further away geometry is represented more coarsely than nearby features; (2) it can be *stored compactly* using standard image coding techniques and tiled for network delivery; (3) the two layers provide the ability to represent color and geometric *details at disocclusions*; (4) the *connectivity* enables converting the 3D photo into a dense mesh that can be rendered without gaps and easily simplified for low-end devices using standard techniques.

Our 3D photo reconstruction algorithm starts from a set of casually captured photos (Section 3.2.1, Figure 3.2a) and uses an existing structure from motion algorithm to estimate the camera poses and a sparse geometric representation of the scene (Section 3.2.2, Figure 3.2b). Our technical innovations concentrate in the following three stages:

**Dense reconstruction** (Section 3.2.3 / Figure 3.2c)**.** Taking the sparse reconstruction as a starting point, we first compute complete depth maps for the input images. We propose a novel prior, called the *near envelope*, that constrains the depth using a conservatively but tightly estimated lower bound. This prior results in highly improved reconstructions in our setting.

**Parallax-tolerant stitching** (Section 3.2.4.1-2 / Figure 3.2d-e)**.** The next goal is to merge the depth images into the final representation. We forward-warp the depth images into the panoramic domain, for each image generating a *front warp* using a standard depth test and a *back warp* using an inverted depth test.

**Two-Layer Fusion** (Section 3.2.4.3 / Figure 3.2f)**.** Next, we merge these images into a front and back stitch, respectively, by solving a MRF problem. These two stitches are finally combined into the two-layer 3D photo representation using an algorithm that resolves connectivity among the layers, removes redundancies, and hallucinates new color and depth data in unobserved areas.

## 3.2 3D Photo Reconstruction

### 3.2.1 Capture and Pre-processing

Most of our scenes were captured with a mid-range Canon EOS 6D DSLR with a 180° fisheye lens. To achieve nearly full $360 \times 180°$ coverage, we captured two rings while pointing the camera slightly up and down, respectively. We fixed the exposure and captured about 25 input images on each ring. We preprocessed the RAW images in Adobe Lightroom to automatically white balance, denoise, and remove color fringing. We also cropped out the circular image region in the fisheye images, leaving about $160^\circ \times 107^\circ$ field-of-view (FOV).

We also experimented with capturing with a Samsung Galaxy S7 smart phone. Since the field-of-view is significantly narrower, we only captured partial panoramas, taking about $4 \times 4$ images on a grid. We used a single fixed exposure for each cell phone scene. This type of capture takes on the order of 30 seconds.

### 3.2.2 Sparse Reconstruction

We used the COLMAP structure from motion package [52] to recover the intrinsic and extrinsic camera parameters, the pose for each image, and a sparse point cloud representation of the scene geometry. We left most COLMAP options at their default, except for the camera model and initial focal length. COLMAP also outputs undistorted rectilinear images of about $110^\circ \times 85^\circ$ ($1350 \times 900$) and $65^\circ \times 50^\circ$ ($1170 \times 900$) FOV (resolution) for the DSLR and cell phone camera, respectively, which we used for all subsequent processing steps.

### 3.2.3 Dense Reconstruction

Our first goal is to densify the reconstruction by computing a dense depth map for each input image using MVS. Unfortunately, our capture process breaks many common assumptions of these algorithms: (1) our baseline is narrow compared to the scene scale, making estimation of far geometry unreliable; (2) many of our scenes are not fully static and contain, for example, swaying trees and moving cars; (3) many scenes contain large textureless regions, such as sky or walls. These issues make MVS unreliable in the affected areas. In fact, most algorithms use some internal measure of confidence and drop pixels that are deemed unreliable, resulting in incomplete reconstructions. If they are forced to return a complete depth map, however, it is usually erroneous and/or noisy in those areas.

We solve this problem by introducing a novel *near envelope* reconstruction prior. The idea behind the prior is to propagate a conservative lower depth bound from confident to the less confident pixels. The prior effectively discards a large fraction of erroneous near-depth hypotheses from the cost volume, and causes the optimizer to reach a better solution.

#### 3.2.3.1 Plane-sweep MVS Baseline

The baseline for our near envelope prior results is a state of the art plane-sweep MVS algorithm, which we describe briefly in this section. Please refer to Appendix B for full implementation details.

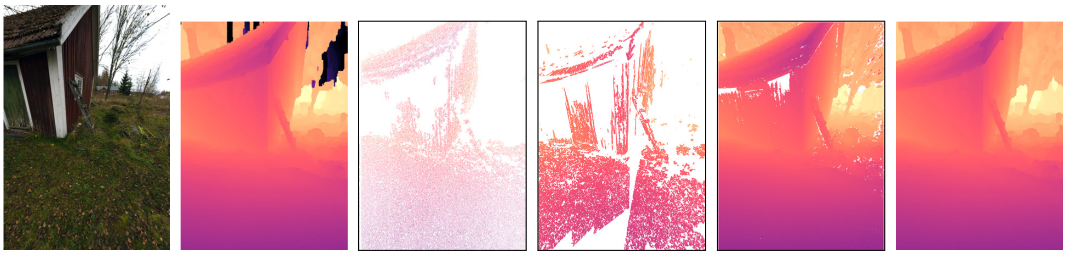

(a) Input view (b) MRF (c) PMVS [16] (d) MVE [89] (e) COLMAP [17] (f) MRF + NE (ours)

**Figure 3.3:** Existing state-of-the-art MVS algorithms (b-e) often produce incomplete and noisy depth maps for our casually captured data. Injecting our near-envelope prior into the MRF baseline algorithm shown in (b) produces superior results (f).

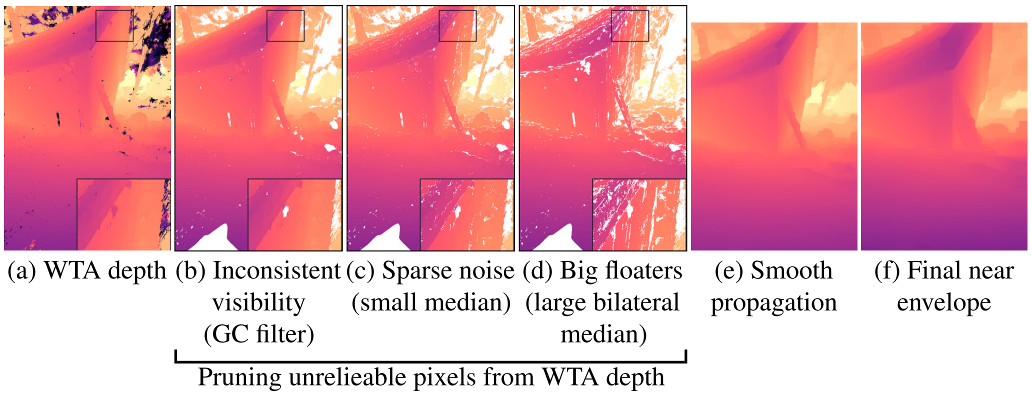

(a) WTA depth
(b) Inconsistent visibility (GC filter)
(c) Sparse noise (small median)
(d) Big floaters (large bilateral median)
(e) Smooth propagation
(f) Final near envelope
Pruning unrelieable pixels from WTA depth

**Figure 3.4:** The near envelope construction starts with a cheap to compute winner-takes-all stereo result (a). We use a series of filters (see text for details) to prune outliers. We fill the gaps that have formed by propagating the depths of the reliable pixels (e). The final envelope is obtained by applying a min-filter and scaling the result (f).

Like prior work [144], we treat depth estimation as an energy minimization problem. Let $i$ be a pixel and $c_i$ its color. We optimize the pixel depths $d_i$ by solving the following problem,

$$\underset{d}{\operatorname{argmin}} \sum_i E_{data}(i) \; + \; \lambda_{smooth} \sum_{(i,j)\in\mathcal{N}} E_{smooth}(i,j) \,, \tag{3.1}$$

which consists of a unary data term and a pairwise smoothness term defined on a four-connected grid, $\lambda_{smooth} = 0.25$ balances their contributions. The smoothness term is the product of a color- and a depth-difference cost,

$$E_{smooth}(i,j) = w_{color}(c_i, c_j)\, w_{depth}(d_i, d_j) \,, \tag{3.2}$$

that encourages the depth map to be smooth wherever the image lacks texture (refer to the supplementary document for details). Our baseline data term consists only of a confidence-weighted photo-consistency term,

$$E_{data}(i) \;=\; w_{photo}(i)\, E_{photo}(i)\,, \tag{3.3}$$

which measures the agreement in appearance between a pixel and its projection into multiple other images (full details for all terms are provided in the Appendix B).

We discretize the potential depth labels and use the plane-sweep stereo algorithm [145] to build a cost volume with 220 depth hypotheses for each pixel. While this restricts us to reconstructing discrete depths without normals, it has the advantage that we can extract a globally optimized solution using an MRF solver, which can often recover plausible depth for textureless regions using its smoothness term. We optimize the MRF at a reduced resolution using the FastPD library [146] for performance reasons. We then upscale the result to full resolution with a joint bilateral upsampling filter [147], using a weighted median filter [67] instead of averaging to prevent introducing erroneous middle values at depths discontinuities.

#### 3.2.3.2 Near Envelope

As mentioned before, most existing MVS algorithms, including the one described in Section 3.2.3.1, do not produce good results on our data. We have tried a variety of available commercial and academic algorithms (Section 3.3.2); Figure 3.3 and Figure 3.5b provide representative results. As can be seen, near-depth hypotheses are noisy, because these points are seen in fewer images and the photo-consistency measure is therefore less reliable. This makes MVS algorithms more likely to fall victim to common stereo pitfalls, such as: repeated structures in the scene, slight scene motion, or materials with view-dependent (shiny) appearance.

The idea behind the near envelope is to estimate a conservative but tight lower bound $n_i$ for the pixel depths at each pixel. We use this boundary to discourage nearby erroneous depths by augmenting the data term in Equation 3.3 with an additional cost term,

$$E_{near}(i) = \begin{cases} \lambda_{near} & \text{if } d_i < n_i \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

that penalizes reconstructing depths closer than the near envelope ($\lambda_{near} = 1$). The near envelope effectively prunes a large fraction of the cost volume, which makes it easier to extract a good solution.

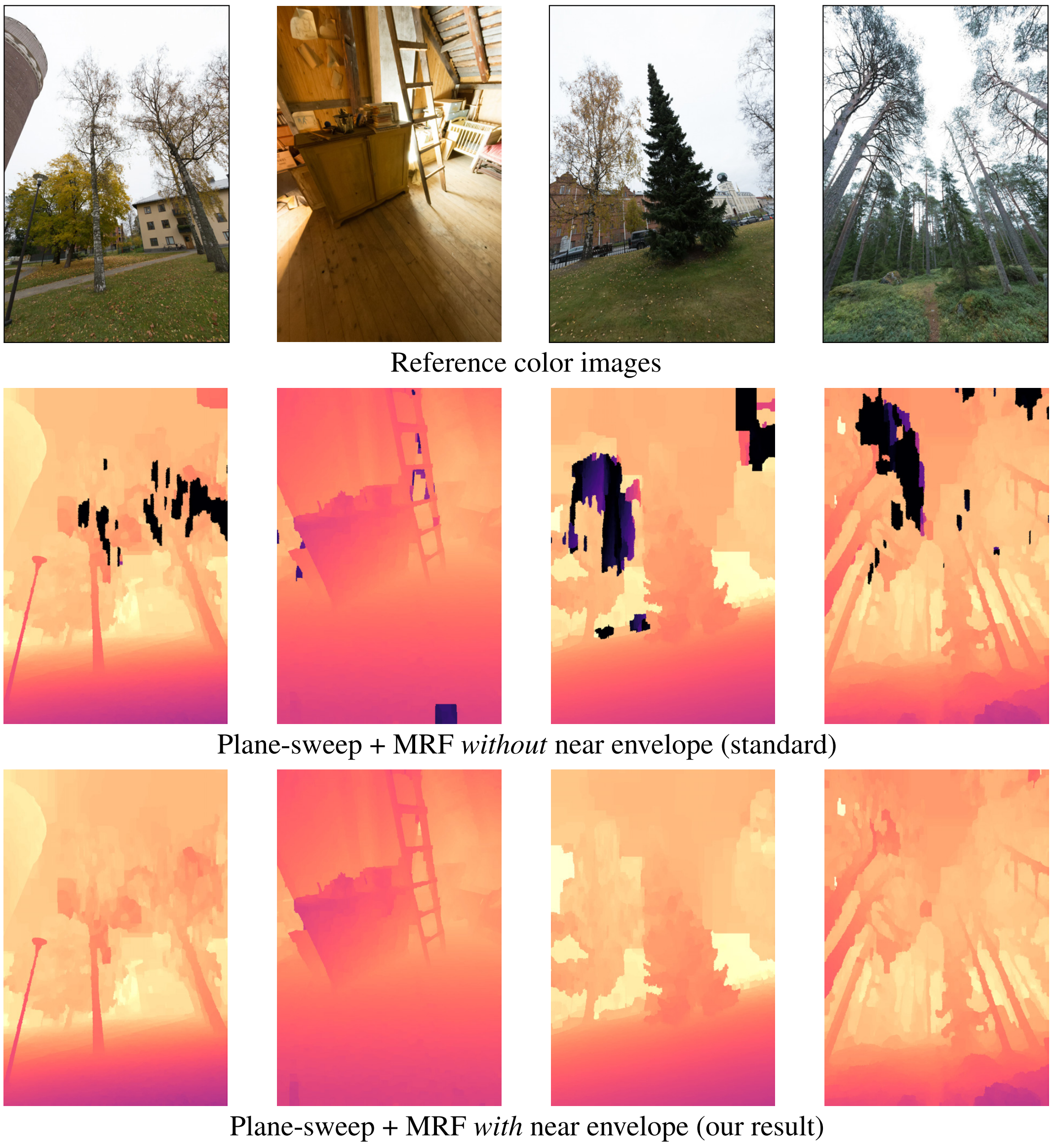

Reference color images
Plane-sweep + MRF *without* near envelope (standard)
Plane-sweep + MRF *with* near envelope (our result)

**Figure 3.5:** Comparison of plane sweep stereo without and with near envelope. Note the erroneous near-depths (dark colors) in the middle row. The supplementary material contains complete depth map result sets for all of our scenes.

To compute the near envelope, we first identify pixels with reliable depth estimates to serve as anchors (Figure 3.4a-d). We propagate their depths to the remaining pixels using a color affinity smoothness term (Figure 3.4e), and obtain the near envelope by filtering the map and pulling it forward in depth (Figure 3.4f).

**Computing the anchor pixels:** We start by computing a cheap "winner-takes-all" (WTA) stereo result (Figure 3.4a) by dropping the smoothness term from Equation 3.1 and independently computing the minimum of $E_{data}$ for each pixel. This quantity can be computed very fast, but the resulting depth maps are very noisy and contain a mixture of reliable and

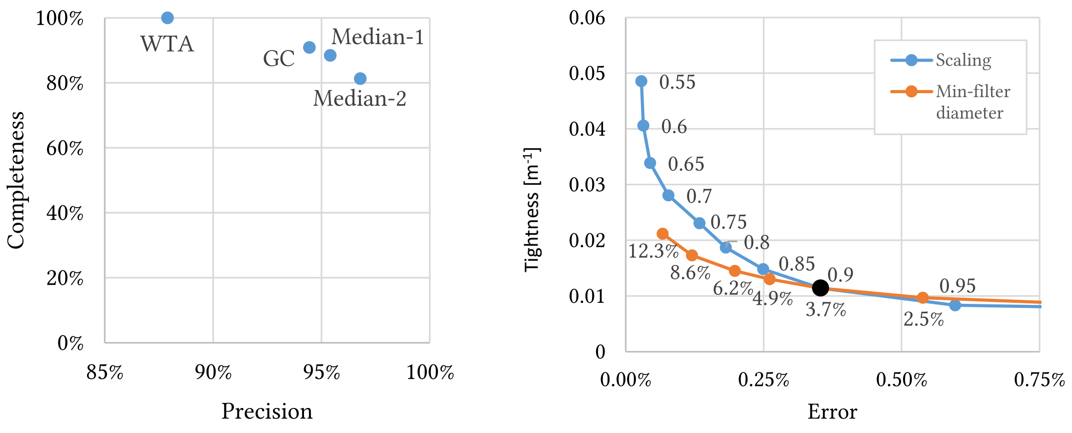


**Figure 3.6:** Evaluating the near envelope construction using a set of ground truth depth maps from various scenes. *Left:* Completeness and precision plot for the outlier pruning stages. *Right:* Tightness and Error plots for the final near envelope. Please see the text for details.

outlier depths (Figure 3.4a). It is critical that the near envelope only be constructed from reliable pixels, otherwise it might either become ineffective due to being too lenient, or it might erroneously cut off true nearby objects, which then cannot be recovered by the final depth computation. Therefore, we wish to filter the outliers from the WTA depth maps. This is difficult, however, because we are faced with two conflicting objectives: (1) we want the filter to be *conservative* so it only lets through reliable pixels, but on the other hand (2) we need the filtered depth maps to be as *complete* as possible so we do not remove truly existing content.

We resolve this problem by applying a judiciously selected combination of pruning filters (Figure 3.4b-3.4d). Each filter is focused on a different kind of outlier, but they are all designed to preserve nearly all reliable pixels, so that their combination achieves both objectives stated above.

The first step is to evaluate the geometric consistency (GC) measure, (using a depth threshold $\tau_r = 1.33$, a free space threshold $\tau_f = 3$ and consensus threshold $\tau_c = 1$ — with weighted median post-processing, see Section 2.2.4.1 for details). This boils down to examining each 3D point originating from a depth map against all other depth maps and checking whether it is consistent with surfaces implied by the other views. If the point is in conflict with the visibility in other depth maps or not in agreement with a sufficient number of other depth maps it is deemed inconsistent, and we remove the point (Figure 3.4b).

We tune the filter carefully to avoid removing too many inliers. As a result, its output still contains some amount of outlier pixels that fall into two categories: sparse small noise regions (almost like salt-and-pepper noise) and larger floaters. These remaining outliers can be pruned using a median filter: a pixel is pruned if its depth is sufficiently different from the median filtered depth map (i.e., not within a factor of $[0.9, 1.11]$).

Tuning the size of the median filter is problematic. A large filter removes thin structures, while a small filter cannot prune large floaters. Our solution is to split the filter into two steps. We first use a small median ($5\times5$ on a $1350\times900$ image) to prune the sparse noise (Figure 3.4c). For larger outliers we use a much larger $51\times51$ median, but use a bilateral color weight [67] ($\sigma_c = 0.033$) (Figure 3.4d) to prevent the removal of thin structures.

**Propagating the anchor depths:** We spread the sparse anchor depths to the remaining pixels by solving a linear system, similar to the one used in Levin et al.'s colorization algorithm [148],

$$\underset{x}{\operatorname{argmin}} \sum_i w_i \left(x_i - x_i'\right)^2 + \sum_{(i,j)\in\mathcal{N}} w_{ij} \left(x_i - x_j\right)^2, \tag{3.5}$$

where $x_i'$ are the depths of the anchor pixels (where defined), and $x_i$ are the densely propagated depths we solve for. $w_{ij} = e^{-\left(c_i - c_j\right)^2 / 2\sigma_{env}^2}$ is the color based affinity term used by Levin et al. [148] ($\sigma_{env} = 0.1$). $w_i = 250\sigma_i^2$ is a unary term that gives more weight to pixels in textured regions where MVS is known to be more reliable [22], which we set to the variance $\sigma_i^2$ of the color in the surrounding $19 \times 19$ patch.

**Computing the final near envelope:** We make the propagated depths more conservative by multiplying them with a constant factor of 0.9 and subsequently applying a morphological minimum filter with diameter set to about 3% of the image diagonal.

Figure 3.5 compares the MVS algorithm from the previous section with and without the near envelope on a few representative images. In the supplementary material we provide the complete set of images for all scenes, and also a comparison to other MVS algorithms.

**Evaluation:** To verify our near envelope construction, we generated a set of six ground truth depth maps across different scenes. We generated these by keeping only the most confident pixels from a COLMAP 2.1 reconstruction and manually inspected them for correctness.

For the WTA result as well as the three pruning stages (Figures Figure 3.4a-d), we plot the completeness (percentage of pixels retained) as well as the precision (percentage of inliers among the retained pixels) (Figure 3.6, left). The plots show that we retain a large fraction of pixels while eliminating nearly all outliers.

For the final near envelope (Figure 3.4f), we evaluate the error (percentage of pixels where near envelope cuts actual content) as well as the tightness (average inverse distance

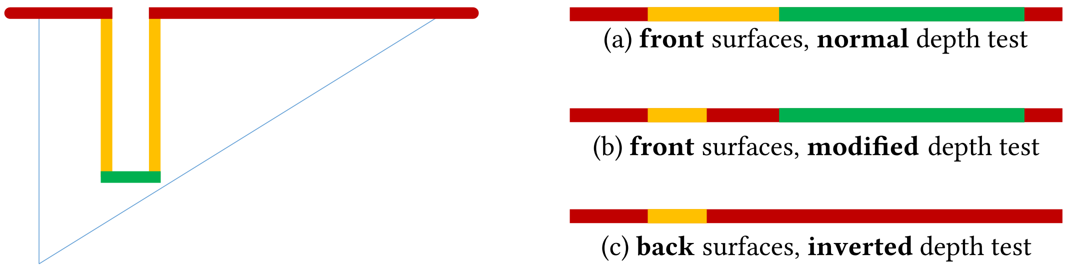

(a) **front** surfaces, **normal** depth test
(b) **front** surfaces, **modified** depth test
(c) **back** surfaces, **inverted** depth test

**Figure 3.7:** We warp the depth maps into the panorama by rendering them with a special depth test to generate front and back surfaces for stitching. (a) highly stretched triangles at depth discontinuities can obscure other good content. (b) We use a modified depth test (see text) to prefer non-stretched triangles even if they are further away. (c) We obtain back surface warps by inverting the depth test.

between the near envelope and actual content). The sweeps for the min-filter size and scaling factor parameters in Figure 3.6, right show that our default parameter (black dot) strikes a good balance between error and tightness.

### 3.2.4 Parallax-tolerant Stitching and Two-layer Fusion

In this section, we merge the depth maps into a panorama. Image alignment and stitching is a well studied problem [64], but most methods assume that there is little parallax between the images. There are MVS algorithms that fuse depth maps (or directly compute a fused result) [25], but their output is not optimized for a specific viewpoint.

Our algorithm starts by warping all the depth maps into the panoramic domain (Section 3.2.4.1), then stitches a front surface and a back surface panorama (Section 3.2.4.2), and finally merges the two into a single two-layered mesh (Section 3.2.4.3).

#### 3.2.4.1 Front and Back Surface Warping

We warp each depth map into a central panoramic image (using equirectangular projection) by triangulating it into a grid mesh and rendering it with the normal rasterization pipeline, letting the depth test select the front surface when multiple points fall onto the same panorama pixel. One problem with this simple approach is that long stretched triangles at depth discontinuities might obscure other good content, and we do not want to include them in the stitch.

We resolve this problem by blending the z-values of rasterized fragments with a stretch penalty $s \in [0,1]$ before the depth test, $z' = (z+s)/2$. The division by 2 keeps the value $z'$ in normalized clipping space. The stretch penalty,

$$s = 1 - \min\left(\frac{\alpha}{\tau_{stretch}},\, 1\right), \tag{3.6}$$

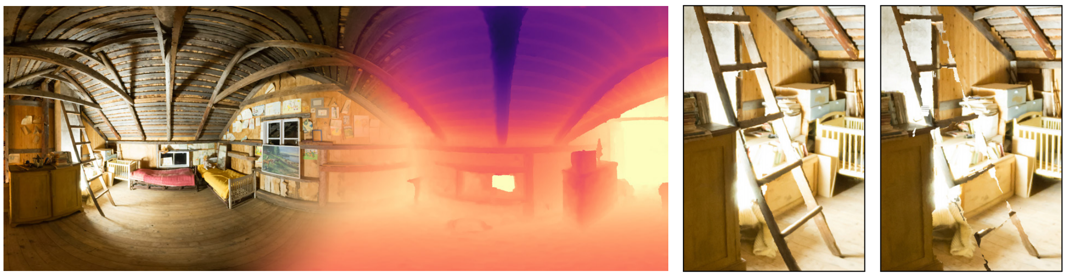

(a) Front color-and-depth panorama (b) Front detail (c) Back detail

**Figure 3.8:** Our algorithm first stitches a color-and-depth panorama for the front-surfaces (a). Its depth is used as constraint, when subsequently stitching the back-surface panorama (c, only detail shown). Note the eroded foreground objects when comparing the back and front details (b-c).

considers the grazing angle $\alpha$ from the original viewpoint and penalizes small values below $\tau_{stretch} = 1.66°$, i.e., rays that are nearly parallel to the triangle surface. This modification pushes highly stretched triangles back, so potentially less stretched back surfaces can win over instead (Figure 3.7b).

Since we are not just interested in only reconstructing the first visible surface, we generate a second *back surface* warp for each image. One possible way to generate this is depth peeling [142]; however, this method is best suited for accurate depth maps and did not perform well on our noisy estimates. We achieved more robust results, instead, by simply inverting the depth test, i.e. $z'' = 1 - z'$. This simple trick works well because when reprojecting depth maps to slightly new viewpoints, the resulting depth complexity rarely exceeds two layers.

#### 3.2.4.2 Single Front and Back Layer Stitching

We are now ready to combine the warped color and depth maps from the previous section into stitched front and back surface panoramas with depth. This involves solving a discrete pixel labeling problem, where each pixel $i$ in the panorama chooses the label $\alpha_i$ from one of the warped sources. We use $c_i^{\alpha_i}$ to denote the color of $i$ from the source image $\alpha_i$ and use an analogous notation for other warped value maps (depth, stretch, etc.)

There are a number of different data and smoothness constraints that have to be considered and are sometimes conflicting. We first stitch the front, and subsequently the back-surface panorama by optimizing the following objective:

$$\begin{aligned}\underset{\{\alpha_i\}}{\operatorname{argmin}} \sum_i \underbrace{\lambda_1\,\Phi_{gc}(i) + \lambda_2\,\Phi_{stretch}(i) + \lambda_3\,\Phi_{depth}(i)}_{\text{Data terms}} + \\ \sum_{i,j\in\mathcal{N}} \underbrace{\lambda_4\,\Psi_{color}(i,j) + \lambda_5\,\Psi_{disp}(i,j)}_{\text{Smoothness terms}}.\end{aligned} \tag{3.7}$$

The geometric consistency term $\Phi_{gc}$ is a binary function set to 1 for pixels that fail the test (using a depth threshold $\tau_r = 1.33$, a free space threshold $\tau_f = 1$ and consensus threshold $\tau_c = 2$ — with weighted median post-processing, see Section 2.2.4.1 for details), and 0 otherwise. The triangle stretch term $\Phi_{stretch}$ discourages selecting pixels from long "rubber sheet" triangles (using the stretch penalty defined in Equation 3.6):

$$\Phi_{stretch}(i) = -\log s_i^{\alpha_i}. \tag{3.8}$$

When stitching front surfaces, we use the depth data terms $\Phi_{depth}$ to encourage depths that are consistent among many views. Similar to the GC filter, we project the 3D point for each pixel into all other depth maps and check whether the projected point's depth and the depth map value are similar. We count the number $n_i$ of other depth maps that are consistent in this manner, and set the depth data term:

$$\Phi_{depth}(i) = 1 - \min\left(\tfrac{n_i}{\tau_{overlap}},\ 1\right). \tag{3.9}$$

$\tau_{overlap} = 5$ specifies how many depth maps need to agree before we consider a pixel to be reliable.

When stitching back surfaces, we use a different depth term that discourages selecting pixels that are closer than the front stitch:

$$\Phi_{depth}(i) = \begin{cases} 10 & \text{if } d_i^{\alpha_i} < 0.95\, d_i^{front} \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

The color smoothness term $\Psi_{color}$ is a truncated version of the seam-hiding pairwise cost from the GraphCut Textures paper [104]:

$$\Psi_{color}(i,j) = \min\left(\|c_i^{\alpha_i} - c_i^{\alpha_j}\|_2^2, \tau_c\right) + \min\left(\|c_j^{\alpha_i} - c_j^{\alpha_j}\|_2^2, \tau_c\right), \tag{3.11}$$

and the depth smoothness term $\Psi_{disp}$ is a similar truncated term (working with disparities):

$$\Psi_{disp}(i,j) = \min\left(\left|\frac{1}{d_i^{\alpha_i}} - \frac{1}{d_i^{\alpha_j}}\right|, \tau_d\right) + \min\left(\left|\frac{1}{d_j^{\alpha_i}} - \frac{1}{d_j^{\alpha_j}}\right|, \tau_d\right). \tag{3.12}$$

The depth $d$ is normalized to metric units, and we use the truncation thresholds $\tau_c = 0.05,\ \tau_d = 0.05\text{m}^{-1}$.

We set the following balancing coefficients,

$$\lambda_1 = 200,\ \lambda_2 = 1000,\ \lambda_3 = 100,\ \lambda_4 = 200,\ \lambda_5 = 100, \tag{3.13}$$

and solve the labeling problems at a reduced $512 \times 256$ resolution (to achieve a reasonable performance) using the alpha expansion algorithm [68]. We then upsample the resulting label map to full resolution ($8192 \times 4096$ pixels) using a simple PatchMatch-based upsampling algorithm [149].

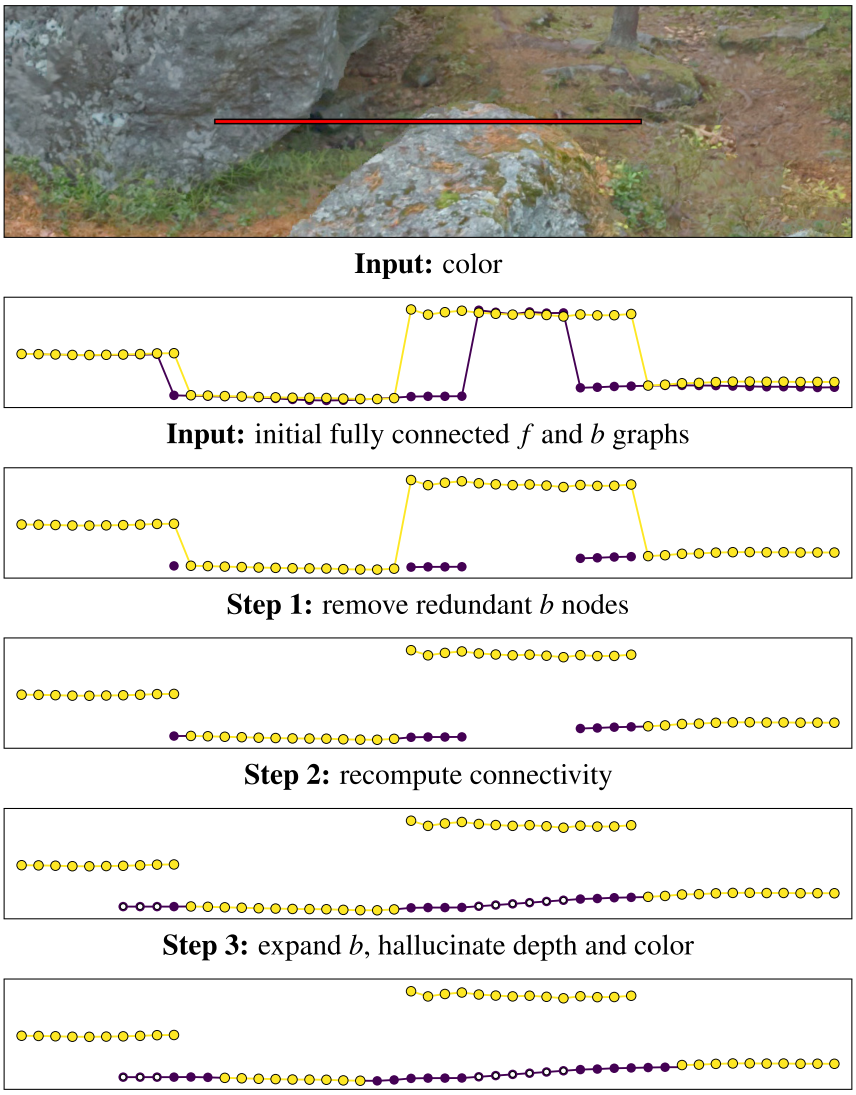
#### 3.2.4.3 Two-layer Merging

Our next goal is to the fuse front and back stitch into the final two-layer representation, to produce a light-weight mesh representation of the scene that is easy to render on a wide variety of devices. Compared to rendering both layers separately, the two-layer fused mesh provides several benefits by (1) resolving which pixels should be connected as part of the same surfaces and where there should be gaps (2) removing color fringes and allows for seamless texture filtering across layers, (3) identifying and discarding redundant parts of the background layer, (4) hallucinating unseen geometry by extending the background layer.

We represent the two-layer panorama as a graph. Each pixel $i$ in the panorama has up to two nodes that represent the foreground and background layers; if they exist, we denote these nodes as $f_i$ and $b_i$. Each node $n$ has a depth value $d(n)$ and a foreground / background label $l(n) \in \{F, B\}$.

We start by generating for both front and back stitches, fully 4-connected but disjoint grid-graphs (Figure 3.9, top). Each node is assigned a depth and label according to the stitch it is drawn from.

Note that these graphs contain redundant coverage of some scene objects, e.g., the inner core of the rocks in Figure 3.9. This is fully intentional and we take advantage of it to remove color fringes around depth continuities further below. However, for now, we remove the redundancies by removing all the $b_i$ nodes that are too similar to their $f_i$ counterparts, i.e.,



**Input:** color

**Input:** initial fully connected $f$ and $b$ graphs

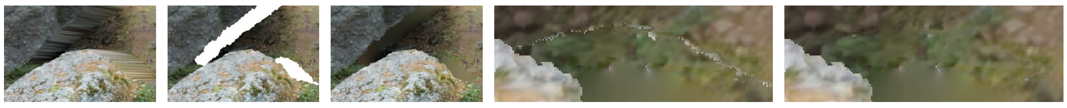
**Step 1:** remove redundant $b$ nodes

**Step 2:** recompute connectivity

**Step 3:** expand $b$, hallucinate depth and color

**Step 4:** propagate $B$ label onto $f$ nodes to remove color fringes

**Figure 3.9:** Fusing the front and back stitches into a single two-layer graph representation. The diagrams show the front and back nodes for the highlighted scanline. Nodes with $F$ and $B$ labels are drawn in yellow and purple, respectively. Nodes with hallucinated depth and color are outlined.



Initial graph | Before expansion | After expansion | Before color fringe removal | Final result

**Figure 3.10:** Steps of the two-layer fusion algorithm. Compare with the diagrams in Figure 3.9.

$d(f_i)/d(b_i) < \tau_{dratio} = 0.75$ (step 1 in Figure 3.9).

The graphs are not redundant anymore, but now the $b$ graph contains many isolated components, and the $f$ graph contains long connections across discontinuities. We recompute the connectivity (step 2 in Figure 3.9) as follows. For each pair of horizontally or vertically neighboring pixels we consider all combinations of $f$ and $b$ nodes and sort them by their depth ratio, most similar first. Then we connect the most similar pair if the depth ratio is above $\tau_{dratio}$. If there is another pair that can be connected without intersecting with the previous one, we connect that as well. Now we have a well-connected two-layer graph.

For small viewpoint changes, the back layer provides some extra content to fill gaps, but large translations can still reveal holes. To improve this, we expand the back layer by hallucinating depth and color (step 3 in Figure 3.9) in an iterative fashion, one pixel ring at a time. At each iteration, we identify $b$ and $f$ nodes that are not connected in one direction. We tentatively create a new candidate neighbor node for these pixels and set their depths and colors to the average of the nodes that spawned them. We keep candidate nodes only if they are not colliding with already existing nodes (using $\tau_{dratio}$), and if they become connected to the nodes that spawned them.

## 3.3 Results and Evaluation

We have captured and reconstructed a number of 3D photos that can be seen in Figure 3.11, as well as the accompanying video and supplementary material. 14 of these scenes were captured with a DSLR camera and 5 with cell phone cameras, as described in Section 3.2.1. The scenes span man-made and natural environments, indoors and outdoors, as well as full $360^\circ \times 180^\circ$ and partial coverage. They contain many elements that are difficult to reconstruct, such as water surfaces, swaying trees, and moving cars and people.

Like other end-to-end reconstruction systems, our system depends on many parameters. We tuned each component individually, following the data flow. For each component, we identified opposing artifacts in a subset of the scenes (e.g., thin objects vs. textureless areas), and performed a parameter sweep to find values that balance both kinds of artifacts. All results provided in this chapter and supplementary material and video were created using the same parameter settings, except lens calibration.

### 3.3.1 Performance

All of the 3D photos were reconstructed on a single 6-Core Intel Xeon PC with an NVIDIA Titan X GPU and 256 GB of memory. Table 4.1 lists timings for a representative DSLR capture with 54 input images at $1350 \times 900$ and an output color-and-depth panorama size $8192 \times 4096$ pixels.

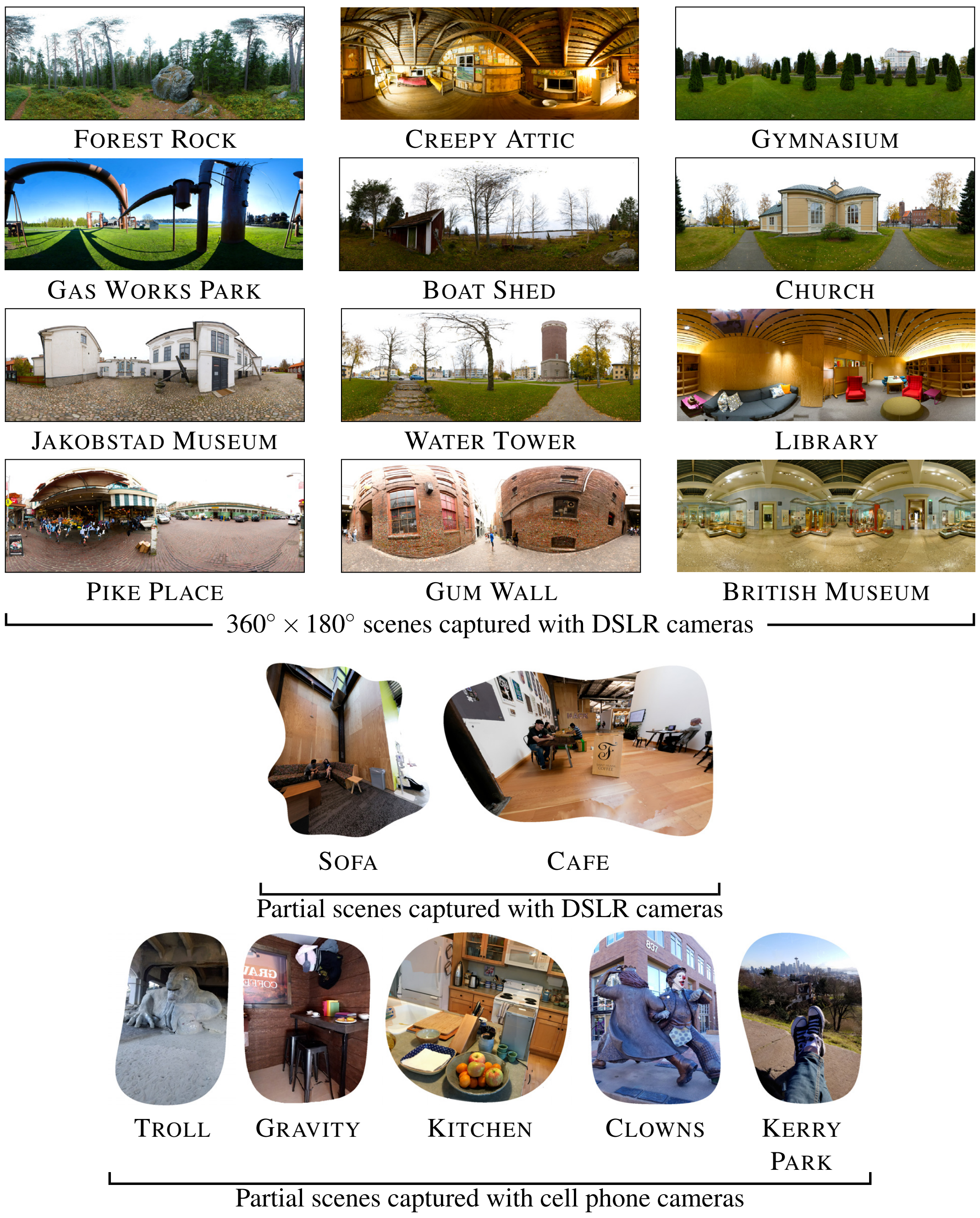


**Figure 3.11:** Some 3D photos we have captured with DSLR and phone cameras.

While our current implementation is slow, we note that there is significant room for improvement. The two bottlenecks are the sparse and the dense scene reconstruction steps, and each can be sped up significantly. A SLAM algorithm would provide a much faster alternative to the slow structure from motion (SfM) algorithm, and it could even improve the results since SLAM is optimized for sequentially captured images while SfM is designed for unstructured wide-baseline input. The MVS reconstruction is by far the slowest step in our

**Table 3.1:** Approximate timings for the 3D photo reconstruction algorithm stages for a whole scene in h:mm.

| **Stage** | **Timing** |
|---|---|
| Sparse reconstruction (Section 3.2.2) | 0:40 |
| Dense reconstruction (Section 3.2.3) | 3:00 |
| Front and Back Warping (Section 3.2.4.1) | 0:15 |
| Front and Back Stitching (Section 3.2.4.2) | 0:30 |
| Two-layer Fusion (Section 3.2.4.3) | 0:30 |
| **Total** | **4:55** |

pipeline, but it could be trivially parallelized, since each image is reconstructed independently from the others.

### 3.3.2 Comparative Evaluation

We have made extensive quantitative and qualitative experiments comparing our system with the following academic and commercial end-to-end reconstruction systems:

**PMVS:** We reconstruct a semi-dense point cloud with PMVS [16] and then use Screened Poisson Surface Reconstruction [96] to create a watertight surface.

**MVE:** We use the Multi-View Environment [89] implementations of Goesele et al.'s [39] semi-dense reconstruction and Floating Scale Surface Reconstruction for texturing [150].

**GDMR:** The Global, Dense Multiscale Reconstruction method [151] starts from the same semi-dense reconstruction as MVE, but provides an alternative surface reconstruction and texturing method.

**COLMAP:** COLMAP 2.1 [52, 17] provides an end-to-end pipeline for sparse reconstruction, depth map computation, and fusion.

**PhotoScan:** Agisoft PhotoScan[1] is a commercial end-to-end reconstruction pipeline.

**Capturing Reality:** This[2] is another state-of-the-art end-to-end reconstruction pipeline.

We use the publicly available implementations for each system. All systems have in common that they initially perform or require as input a sparse reconstruction, which is subsequently densified. To provide a fair comparison, we used the *same* sparse reconstruction (computed with COLMAP) for all systems, including ours.

[1]`http://www.agisoft.com/`
[2]`https://www.capturingreality.com/`

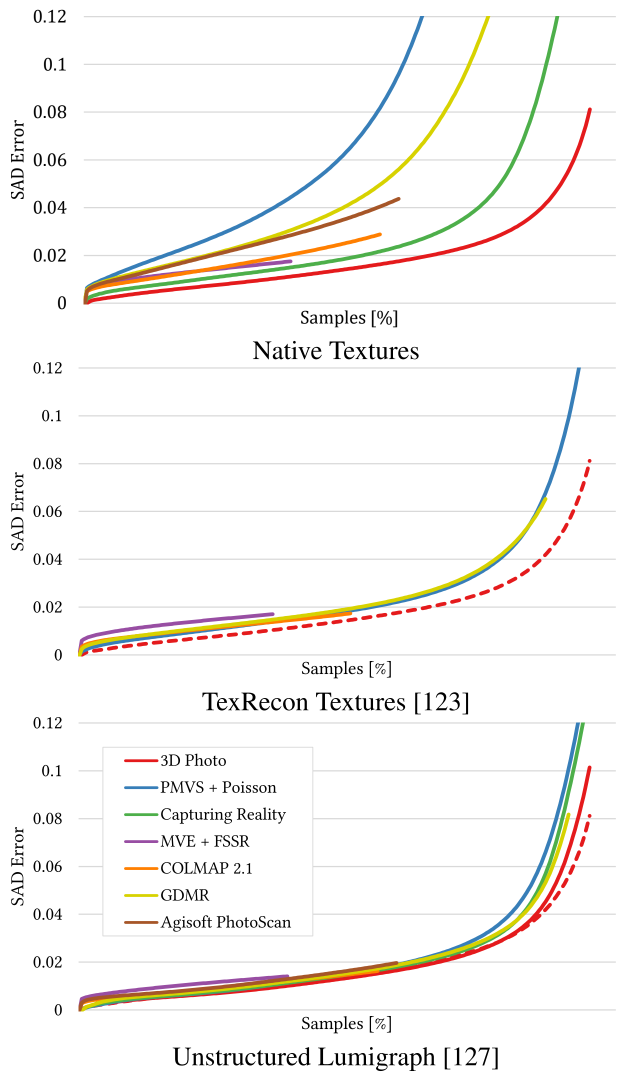
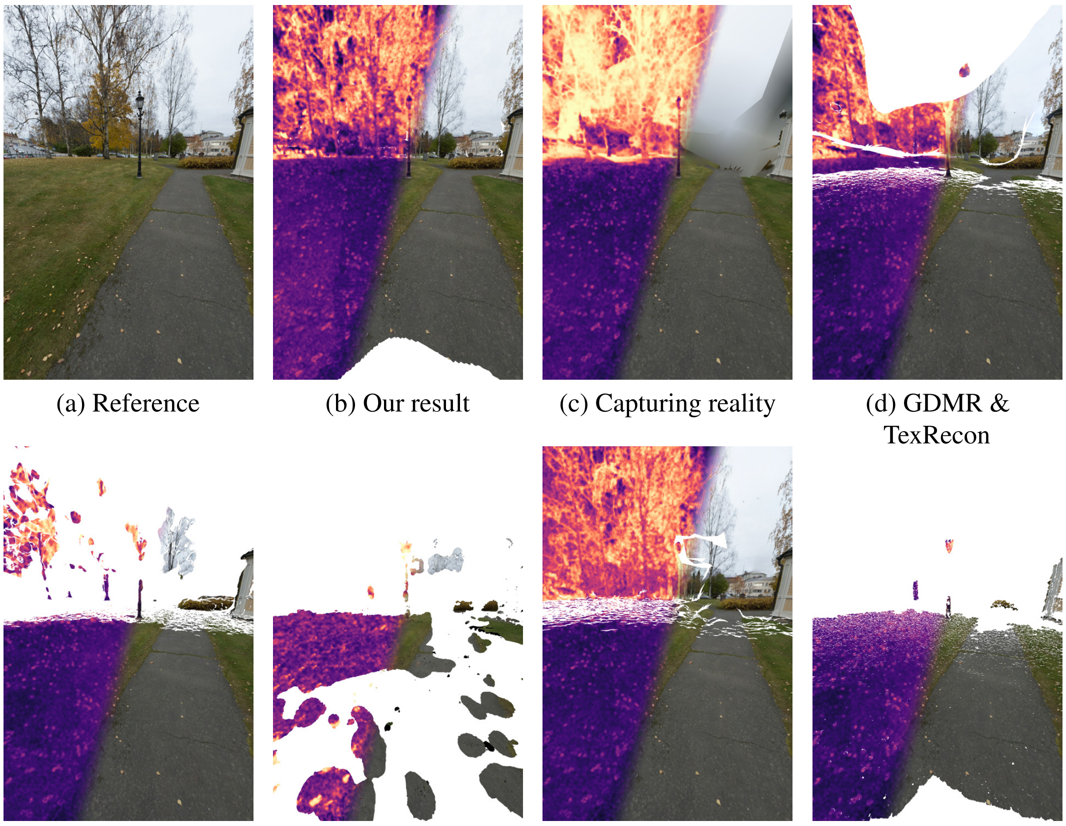


**Figure 3.12:** Plotting the average virtual rephotography error against completeness for different reconstruction and texturing methods. The dotted red line represents our full system with native textures, i.e., is identical to the solid line in the top plot.

Some of the methods produce impressive geometric reconstructions but only relatively coarse textures. For this reason we experimented with the following alternative texturing methods, in addition to whatever native textures each system produces:

**TexRecon** [123]: This method produces a high-quality texture atlas for a given 3D model and images that are registered against this model.

**Unstructured Lumigraph Rendering** [127]: This method uses the 3D model as a geometric proxy for Image-Based Rendering. For each pixel we blend the two top-scoring images to avoid ghosting.


(a) Reference (b) Our result (c) Capturing reality (d) GDMR & TexRecon

(e) COLMAP 2.1 & TexRecon (f) Photoscan (g) PMVS & TexRecon (h) MVE & TexRecon

**Figure 3.13:** Rendered results and corresponding rephotography errors in the Church scene. Dark regions have lower error.

### 3.3.3 Quantitative Evaluation

We use Virtual Rephotography [141] to provide a quantitative comparison against the systems mentioned before. This is a unified evaluation method for end-to-end reconstruction-and-rendering methods. Its idea is to render the reconstruction from the exact same viewpoint as each input image (“virtual rephotos”), and compare the results against the ground truth images with respect to completeness and visual error.

Since the texture resolution of the methods differ, we render results at a low resolution (900 pixels width). For each pixel, we evaluate the minimum sum of absolute differences (SAD) error within a shiftable window of $\pm 2$ pixels.

Figure 3.12 provides error and completeness scores aggregated over all 15 scenes for each reconstruction and the three alternative ways of texturing them, and Figure 3.13 provides a visual result for an example image. In the supplementary material, we provide a more fine-grained break-down of these results.

### 3.3.4 Qualitative Evaluation

To help the reader evaluate the visual quality of the results, we provide a full set of videos with scripted camera paths for all scenes and all methods in the supplementary material, and a web page for convenient side-by-side visual inspection.

A major limitation of most systems we compared against is that they do not produce complete results. Most methods do not reconstruct far regions, due to reliability thresholds in the depth estimation. Another source of missing regions are texture-less areas, where matching is ambiguous. Our technique produces far more complete results and relies on MRF-based smoothing to fill in unreliably or ambiguous regions.

MRF smoothing also helps in regions with highly complex geometry, where systems without smoothing (e.g., COLMAP 2.1) produce noisy results.

Some of the systems produce only relatively coarse vertex color textures. TexRecon [123] does a good job in improving the texturing, at the expense of making the results a bit less complete, since it removes uncertain triangles. Another alternative is image-based rendering [127]. While IBR is very good at optimizing the reprojection error, it has several downsides: the display is less stable as texturing is view-dependent, leading to visibly moving texture seams when moving the camera. Another disadvantage of IBR is that it is far more expensive than native texturing in terms of data size (all source images need to be retrieved and retained in memory) as well as performance (shading).

### 3.3.5 Stitching Evaluation

In Figure 3.14, we show the effect of running our system without the near envelope and without the GC filter (see the supplementary material for more results). The near envelope provides a substantial improvement in many scenes. Without it, we often see many floaters (dark spots in Figure 3.14a) that lead to visual artifacts (Figure 3.14b). Disabling the GC filter yields more edge fattening in many scenes, though its impact is less dramatic than the near envelope.

### 3.3.6 Limitations

Our system has some important limitations, which are inherited from the underlying sparse and dense reconstruction algorithms. The stereo algorithm may fail to reconstruct shiny and reflective surfaces (e.g., `British Museum`), as well as dynamically moving objects such as people (e.g., `Pike Place, Gum Wall`). This might might lead to erroneously estimated depth. Often, our subsequent processing stages can fix these problems, e.g., the

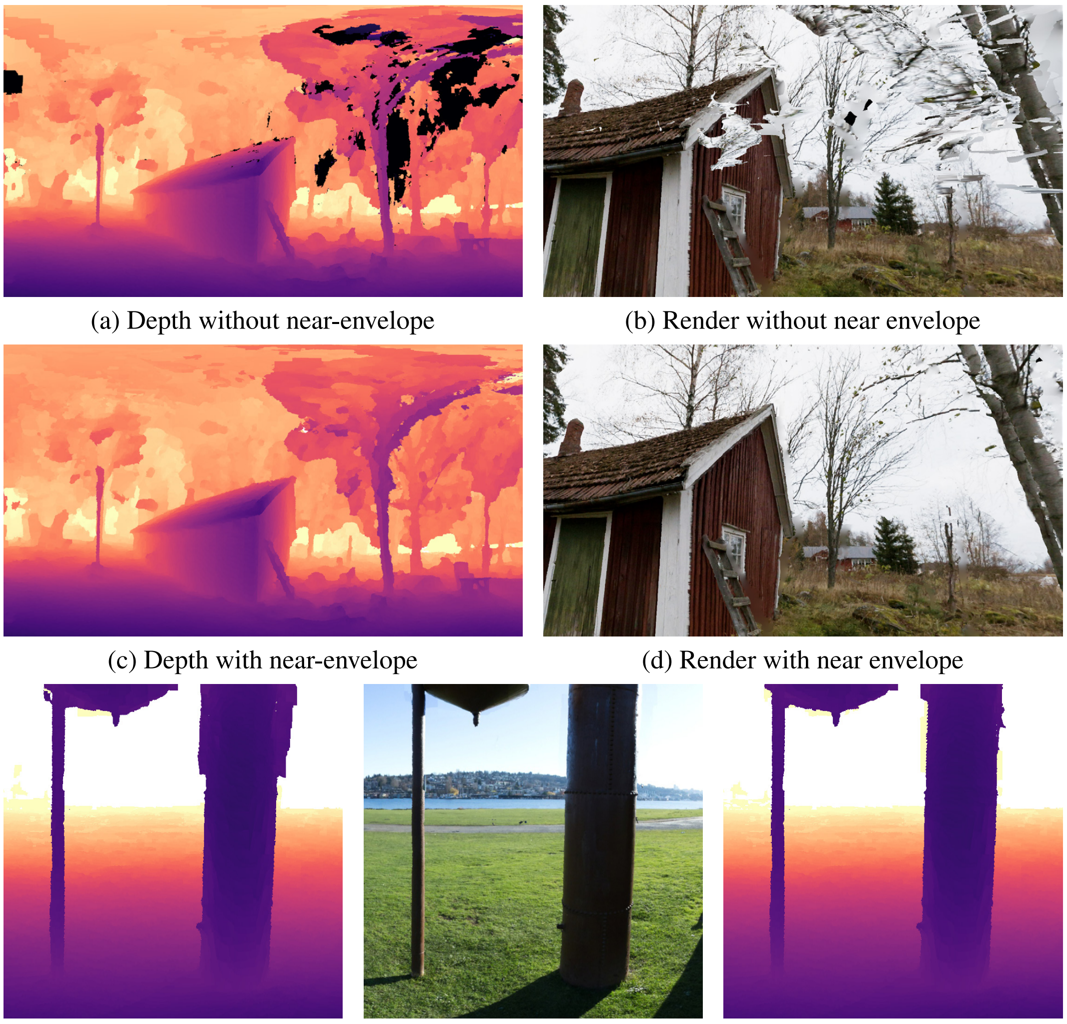

(a) Depth without near-envelope
(b) Render without near envelope
(c) Depth with near-envelope
(d) Render with near envelope
(e) Stitched without GC cost
(f) Stitched colors
(g) Stitched with GC cost

**Figure 3.14:** Showing the effect of running our system without the near envelope prior (a-d), as well as without GC cost in the stitcher (e-g). Running without near envelope yields many floaters that result in visual artifacts. The effect of disabling the GC cost is more subtle, it results in more edge fattening.

stitching step deals well with geometric outliers in single depth maps. But in some of our results, small artifacts can still be found under close inspection (see supplemental videos). In partially captured scenes some artifacts may appear near boundaries where fewer input images observed the scene.

Another minor limitation is that our current implementation does not perform any blending, which sometimes leads to visible color discontinuities (e.g., `Library`, `Kitchen`). It would be relatively straight-forward to add color blending and even depth-blending [114] to our system.

## 3.4 Conclusions

In this chapter, we developed a novel system that constructs a *3D photo*, a seamless two-layer representation for viewpoint-free photos from sequences of casually acquired input photos.

Our work builds on a strong foundation in sparse and dense MVS algorithms, with enhanced results due to our novel near envelope cost volume prior. Our parallax-tolerant stitching algorithm further removes many outlier depth artifacts. It produces front and back surface panoramas with well-reconstructed depth edges, because it starts from depth maps whose edges are aligned to the reference image color edges. Our fusion algorithm fuses the front and back panoramas into a single two-layer 3D photo.

All these technical innovations bring us closer to our goal of easy capture (see Section 1.1): We can apply our algorithm to *casually* captured input images that violate many common assumptions of MVS algorithms. Most of our scenes contain a variety of difficult to reconstruct elements, such as dynamic objects (people, swaying trees), shiny materials (lake surface, windows), and textureless surfaces (sky, walls). The fact that we nevertheless succeed in reconstructing relatively artifact-free scenes speaks for the robustness of our approach.

However, the system presented in this chapter is compute-heavy and requires several hours of processing to reconstruct a 3D photo. Faster processing would make capture even easier. If users could see the final 3D photo still on-set, they can spot mistakes and adjust their capture accordingly. In the next chapter, we address this problem and present a much faster method to reconstruct 3D photos.

## Chapter 4

# Instant 3D Photography

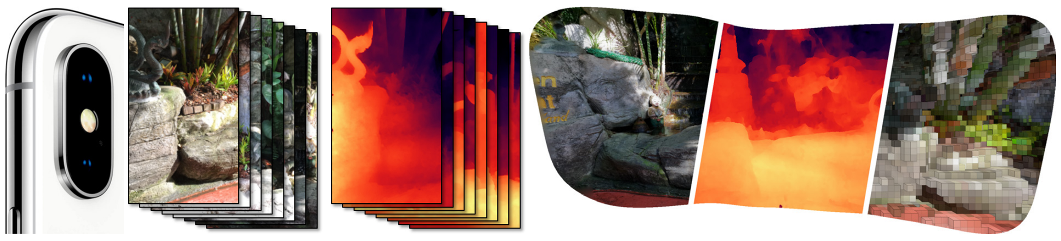

Dual camera phone
Input: 34 color-and-depth photos, captured in 34.0 seconds
Our 3D photo (color, depth, and a 3D effect), generated in 34.7 seconds.

**Figure 4.1:** Our work enables practical and casual free-viewpoint photography with regular dual-camera phones. Like Chapter 3, we target seated VR experiences and use the *3D photo* representation. *Left:* A burst of input color-and-depth image pairs that we captured with a dual camera cell phone at a rate of one image per second. *Right:* 3D photo generated with our algorithm in about the same time it took to capture. The geometry is highly detailed and enables viewing with binocular and motion parallax in VR, as well as applying 3D effects that interact with the scene.

In this chapter, we continue our work on creating viewpoint-free photos for seated VR experiences. Like Chapter 3, we place our focus on easy capture and use the *3D photo* representation: a textured, multi-layered 3D mesh that can be rendered with standard graphics engines. Unlike Chapter 3, the system presented here is specifically designed with processing speed in mind and runs several orders of magnitude faster. The drawback is that, when compared to Chapter 3, this approach has a slightly reduced range-of-motion.

As stated in Section 1.1, we are looking for a method that does not require expensive hardware and is easy to use. Yet, it should create high-quality viewpoint-free photos suitable for seated VR experiences with both binocular vision and head-motion parallax. Finally, in this chapter we focus on fast processing times, on the order of seconds at most.

We present a new algorithm that constructs 3D photos from sequences of color-and-depth photos produced from small-baseline stereo dual camera cell phones, such as recent

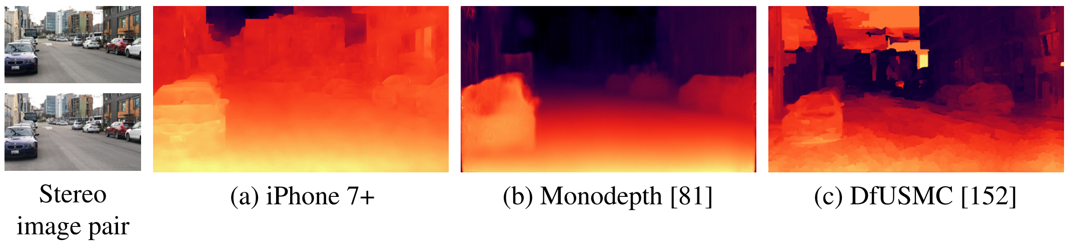


**Figure 4.2:** Estimating depth maps using various algorithms. Note relative scale difference and low-frequency deformations between different maps. (a) Small baseline stereo depth computed by the native iOS algorithm on an iPhone 7+. (b) Single image CNN depth map [81]. (c) Depth from accidental motion result [152] (we actually used a short video clip to produce this result).

iPhones. We take these sequences with a custom burst capture app while casually moving the phone around at a half-arm's distance. The depth reconstruction is essentially free since it is integrated into native phone OS APIs and highly optimized.

In contrast to the approach in Chapter 3, which requires several hours to process a scene, our method is fast and processes approximately one input image per second on a laptop PC, about the same time it takes to capture. We stress the importance of this point, since we found that as our system became faster, it made our own behavior with regards to capture more opportunistic: we were suddenly able to capture spontaneously on the go and even perform multiple captures in the same scene to try different viewing angles.

We demonstrate our algorithm on a wide variety of captured scenes, including indoor, outdoor, urban, and natural environments at day and night time. We also applied our algorithm to several datasets where the depth maps were estimated from single images using CNNs. These depth maps are strongly deformed from their ground truth and lack image-to-image coherence, but nevertheless our algorithm is able to produce surprisingly well-aligned and consistent stitched panoramas. A large number of these results is provided in the supplementary material and accompanying video.

## 4.1 Overview

The goal of our work is to enable easy and rapid capture of 3D photos using readily available consumer hardware.

### 4.1.1 Dual Lens Depth Capture

Dual lens cameras capture synchronized small-baseline stereo image pairs for the purpose of reconstructing an aligned color-and-depth image using depth-from-stereo algorithms [28]. The depth reconstruction is typically implemented in system-level APIs and highly

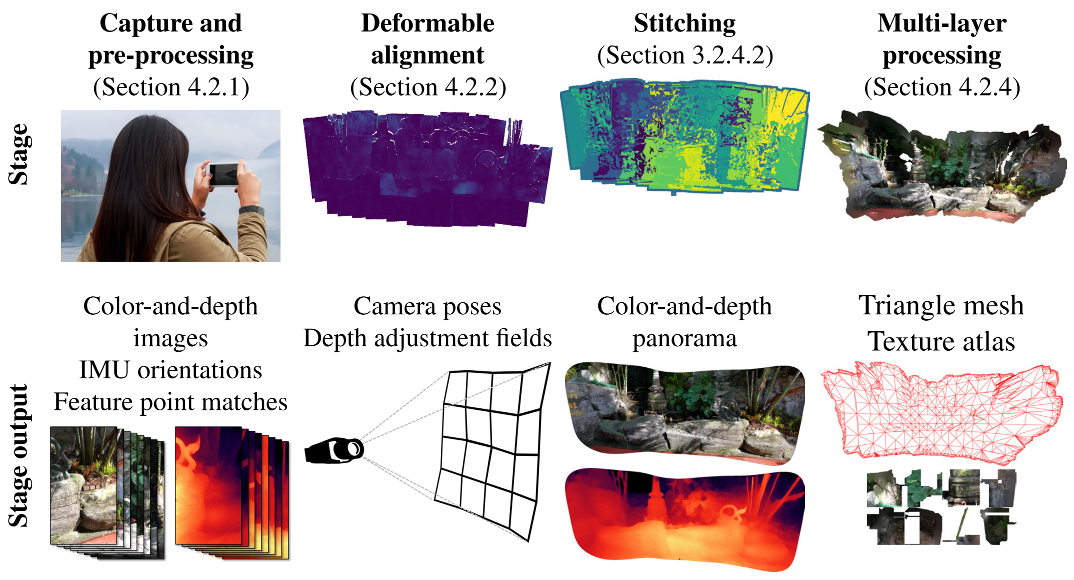


**Figure 4.3:** Breakdown of the major algorithms stages and their outputs, which form the inputs to the next respective stage.

optimized, so from a programmer's and a user's perspective, the phone effectively features a "depth camera". Several recent flagship phones feature dual cameras, including the iPhone 7 Plus, 8 Plus, X, and Samsung Note 8. Such devices are already in the hands of tens of millions of consumers.

The small baseline is both a blessing and a curse: the limited search range enables quickly establishing dense image correspondence but also makes triangulation less reliable and causes large uncertainty in the estimated depth. For this reason, most algorithms employ aggressive edge-aware filtering [153, 154], which yields smoother depth maps with color-aligned edges, but large low-frequency error in the absolute depth values. In addition, the dual lenses on current-generation phones constantly move and rotate during capture due to optical image stabilization, changes in focus, and even gravity[1]. These effects introduce a non-linear and spatially-varying transformation of disparity that adds to the low-frequency error from noise filtering mentioned above.

In Figure 4.2, you can see depth maps reconstructed using different stereo algorithms on this kind of data. As revealed in the figure, there is a significant amount of low-frequency error in the depth maps. Since our focus is not stereo matching, we use the depth maps from the native iPhone 7 Plus stereo algorithm for all of our experiments.

An important detail to note is that many small baseline stereo methods (including the one running on the iPhone) do *not* estimate absolute depth, but instead produce *normalized*

[1]see `http://developer.apple.com/videos/play/wwdc2017/507` at 17:20-20:50, Slides 81-89.

depth maps. So, aligning such depth map involves estimating scale factors for each of them, or, in fact, sometimes even more complicated transformations.

### 4.1.2 Algorithm Overview

Our 3D panorama construction algorithm proceeds in four stages:

**Capture (Section 4.2.1, Figure 4.3a):** The input to our algorithm is a sequence of aligned color-and-depth image pairs, which we capture from a single vantage point on a dual lens camera phone using a custom burst capture app.

**Deformable Depth Alignment (Section 4.2.2, Figure 4.3b):** Due to the small camera baseline and resulting triangulation uncertainty, the input depth maps are not very accurate, and it is not possible to align them well using global transformations. We resolve this problem using a novel optimization method that jointly estimates the camera poses as well as spatially-varying adjustment maps that are applied to deform the depth maps and bring them into good alignment.

**Stitching (Section 3.2.4.2, Figure 4.3c):** Next, we stitch the aligned color-and-depth photos into a panoramic mosaic. Usually this is formulated as a labeling problem and solved using discrete optimization methods. However, optimizing label smoothness, e.g., using MRF solvers, is very slow, even when the problem is downscaled. We utilize a carefully designed data term and the high quality of our depth alignment, to replace label smoothness optimization with independently optimizing every pixel after filtering the data term in a depth-guided edge-aware manner. This achieves visually similar results with more than an order of magnitude speedup.

**Multi-layer Mesh Generation (Section 4.2.4, Figure 4.3d):** In the last stage, we convert the panorama into a multi-layered and textured mesh that can be rendered on any device using standard graphics engines. We tear the mesh at strong depth edges and extend the backside into the occluded regions, hallucinating new color and depth values in occluded areas. Finally, we simplify the mesh and compute a texture atlas.

## 4.2 Algorithm

### 4.2.1 Capture and Preprocessing

We perform all of our captures with an iPhone 7 Plus using a custom-built rudimentary capture app. During a scene capture session, it automatically triggers the capture of color-and-depth photos (using the native iOS stereo algorithm) at 1 second intervals.

The capture motion resembles how people capture panoramas today: the camera is pointed outwards while holding the device at half-arms' length and scanning the scene in an arbitrary up-, down-, or sideways motion. Unfortunately, the field-of-view of the iPhone 7 Plus camera is fairly narrow in depth capture mode (37° vertical), so we need to capture more images than we would with other cameras. A typical scene contains between 20 and 200 images.

The captured color and depth images have $720 \times 1280$ pixels and $432 \times 768$ pixels resolution, respectively. We enable the automatic exposure mode to capture more dynamic range of the scene. Along with the color and depth maps, we also record the device orientation estimate provided by the IMU.

**Feature extraction and matching:** As input for the following alignment algorithm, we compute pairwise feature matching using standard methods. We detect Shi-Tomasi corner features [44] in the images, tuned to be separated by at least 1% of the image diagonal. We then compute DAISY descriptors [155] at the feature points. We use the IMU orientation estimate to choose overlapping image pairs, and then compute matches using the FLANN library [156], taking care to discard outliers with a ratio test (threshold = 0.85) and simple geometric filtering, which discards matches whose offset vector deviates too much from the median offset vector (more than 2% of the image diagonal). All this functionality is implemented using OpenCV.

### 4.2.2 Deformable Depth Alignment

Our first goal is to align the depth maps. Since the images were taken from different viewpoints, we cannot deal with this in 2D image space due to parallax. We need to recover the extrinsic camera poses (orientation and location), so that when we project out the depth maps they align in 3D.

#### 4.2.2.1 Rigid alignment:

We achieve this goal by minimizing the distance between reprojected feature point matches. Let $f_A^i$ be a feature point in image $A$ and $M = \{(f_A^i, f_B^i)\}$ the set of all matched pairs. We define a reprojection loss as follows:

$$E_{reprojection} = \sum_{(f_A^i, f_B^i) \in M} \rho\Big( \left\| P_{A \to B}(f_A^i) - f_B^i \right\|_2^2 \Big), \tag{4.1}$$

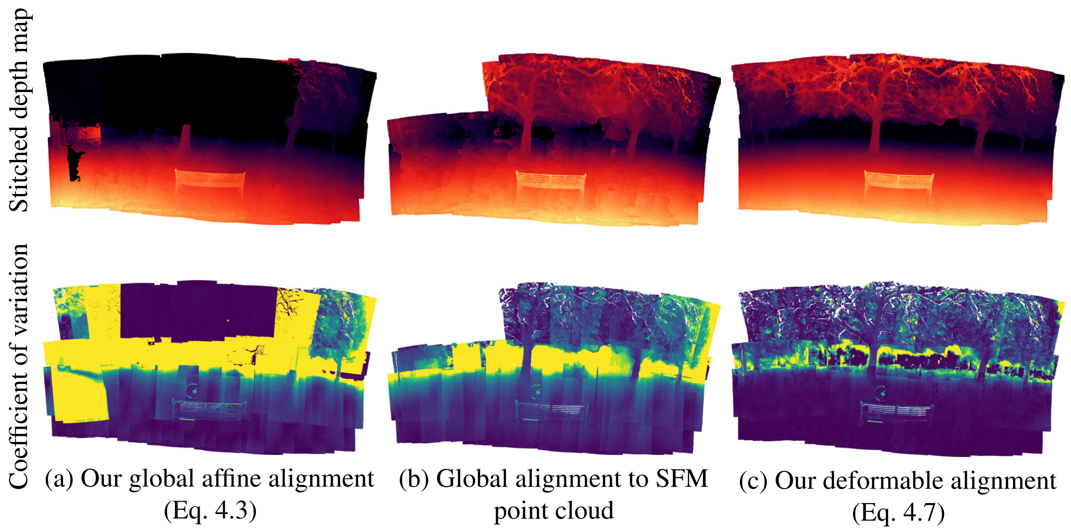


**Figure 4.4:** Aligning depth maps with low-frequency errors. We show stitches and the coefficient of variation (see text) for various methods. (a) Our algorithm with a global affine model (Eq. 4.3). Many depth maps got pushed to infinity. (b) Aligning each depth map *independently* with Eq 4.3 to a high-quality reconstruction. The result is better, but there are many visible seams and floaters due to the impossibility to fit the inaccurate depth maps with simple global transformations. (c) Our algorithm with the spatially-varying affine model yields excellent alignment.

where $\rho(s) = \log(1+s)$ is a robust loss function to reduce sensitivity to outlier matches, and $P_{A \to B}(f)$ is the reprojection function from Section 2.1.3.1, which projects the 2D point $f$ from image $A$ to image $B$, using the extrinsics $(\mathbf{R}_A, \mathbf{t}_A)$ and $(\mathbf{R}_B, \mathbf{t}_B)$ , i. e., the rotation matrix and translation vectors for images $A$ and $B$. Note, that this formulation naturally handles the wrap-around in 360° panoramas.

Similar reprojection losses are common in geometric computer vision and have been used with great success in many recent reconstruction systems [52]. However, our formulation has a subtle but important difference: since we have depth maps, we do not need to optimize the 3D location of feature point correspondences. This significantly simplifies the system in several ways: (1) it drastically reduces the number of variables that need to be estimated, to just the camera poses, (2) we do not have to link feature point matches into long tracks, and (3) the depth maps helps reduce uncertainty, making our system robust to small baselines and narrow triangulation angles.

Eq. 4.1 assumes that the camera intrinsics as well as lens deformation characteristics are known and fixed throughout the capture. If this is not the case, extra per-camera variables could be added to this equation to estimate these values during the optimization.

Minimizing Eq. 4.1 w.r.t. the camera poses is equivalent to optimizing a rigid alignment of the depth maps. However, since most small baseline depth maps are normalized (including the ones produced by the iPhone), they cannot be aligned rigidly.

#### 4.2.2.2 Global transformations:

We resolve this problem by introducing extra variables that describe a global transformation of each depth map. Our first experiment was trying to estimate a scale factor $s_A$ for each depth map, i.e., by replacing $d_A(f)$ in Eq. 4.1 with

$$d_A^{scale}(f) = s_A \, d_A(f), \tag{4.2}$$

where $s_A$ is an extra optimization variable per image. However, this did not achieve good results, because, as we learned, many depth maps are normalized using unknown *curves*. We tried a variety of other classes of global transformations, and achieved the best results with an affine transformation in disparity space (i.e., $1/d$):

$$d_A^{affine}(f) = \Big(s_A \, d_A^{-1}(f) + o_A\Big)^{-1}, \tag{4.3}$$

$s_A$ and $o_A$ are per-image scale and offset coefficients, respectively.

Figure 4.4a shows a typical result of minimizing Eq 4.1 with the affine model. Many depth maps are incorrectly pushed at infinity, because the optimizer could not find a good way to align them otherwise. In the bottom row we visualize the coefficient of variation of depth samples per pixel, i.e., the ratio of the standard deviation to the mean. This is a scale-independent way of visualizing the amount of disagreement in the alignment. As a sanity check we also tried to independently align each depth map to a high quality SfM reconstruction of the scene (computed with COLMAP [52]) that can be considered ground truth (Figure 4.4b). Even with this “best-possible” result for the model the stitch is severely degraded by seams and floaters.

Through our experimentation we found that it is ultimately not possible to bring this kind of depth maps into good alignment using simple global transformations because of to the low frequency error that is characteristic for small baseline stereo depth maps due to the triangulation uncertainty.

#### 4.2.2.3 Deformable alignment:

Our solution to this problem is to estimate spatially-varying adjustment fields that *deform* each depth map and can therefore bring them into much better alignment. We modify the affine model in Eq. 4.3 to replace the global scale and offset coefficients with regular grids of $5 \times 5$ values that are bilinearly interpolated across the image.

$$d_A^{deform}(f) = \left(s_A(f)\, d_A^{-1}(f) + o_A(f)\right)^{-1}, \tag{4.4}$$

where $s_A(f) = \sum_i w_i(f)\, \hat{s}_A^i$, and $o_A(f) = \sum_i w_i(f)\, \hat{o}_A^i$, and $w_i(f)$ are bilinear interpolation weights at position $f$.

To encourage smoothness in the deformation field we add a cost for differences between neighboring grid values:

$$E_{smoothness} = \sum_A \sum_{(i,j)\in N} \|\hat{s}_A^i - \hat{s}_A^j\|_2^2 + \|\hat{o}_A^i - \hat{o}_A^j\|_2^2 \tag{4.5}$$

While $E_{reprojection}$ is agnostic to scale, $E_{smoothness}$ encourages setting the disparity scale functions $\hat{s}_A^i$ very small, which results in extremely large reconstructions. To prevent this, we add a regularization term that keeps the overall scale in the scene constant:

$$E_{scale} = \sum_A \sum_i \left(\hat{s}_A^i\right)^{-1} \tag{4.6}$$

The combined problem that we solve is:

$$\underset{\{\mathbf{R}_I, \mathbf{t}_I, \hat{s}_i, \hat{o}_i\}}{\operatorname{argmin}} E_{reprojection} + \lambda_1 E_{smoothness} + \lambda_2 E_{scale}, \tag{4.7}$$

with the balancing coefficients $\lambda_1 = 10^6$, $\lambda_2 = 10^{-4}$.

Figure 4.4c shows the improvement achieved by using the deformable model. The ground plane is now nearly perfectly smooth, there are no floaters, and thin structures such as the lamp post are resolved much better.

#### 4.2.2.4 Optimization Details:

Since Eq. 4.7 is a non-linear optimization problem, we require a good initialization of the variables. We initialize the camera rotations using the IMU orientations, and the locations by pushing them forward onto the unit sphere, i.e., $\mathbf{t}_A = \mathbf{R}_A \cdot [0,0,1]^\mathsf{T}$. We found it helpful to initialize the deformation field to enlarge the depth maps, i.e., $\hat{s}_A^i = 0.1, \hat{o}_A^i = 0$, because in this way reprojected feature points are visible in their matched images.

We use the Ceres library [157] to solve this nonlinear least-squares minimization

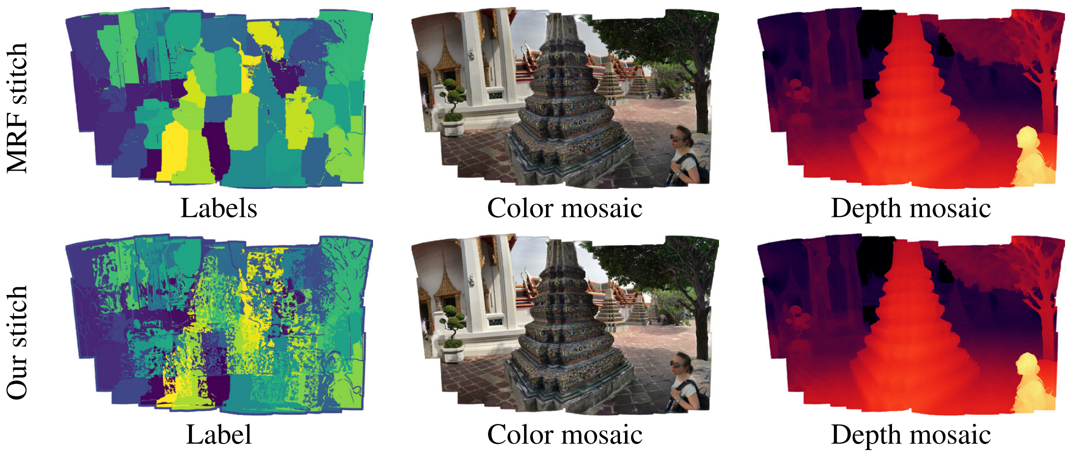


**Figure 4.5:** Comparing stitching using MRF optimization (top row, runtime 3.25 minutes) vs. our algorithm (bottom row, runtime 0.5 seconds). While labels change more frequently in our solution, the color and depth mosaics are visually very similar to the MRF result.

problem using the Levenberg-Marquardt algorithm. We represent all rotations using the 3-dimensional axis-angle parameterization and use Rodrigues' formula [158] when they are applied to vectors. The optimization usually converges within 50 iterations, which takes about 10 seconds.

#### 4.2.2.5 Discussion:

Due to the non-rigid transformations in the optimization the camera poses that we recover are not necessarily accurate anymore. We inspected the recovered poses visually and found that they qualitatively look similar to results obtained by SfM, but we have not performed a careful analysis to verify their degree of accuracy.

We tried also even richer models than Eq. 4.3. In particular we have tried 3D grids that represent *bilateral space* adjustments with depth and luminance as range domains. However, while we found slight improvements in the results we did not deem it significant enough to warrant the extra complexity.

We also experimented with using 3D distance between matched features points' projection into world space in our loss. However, this did not work well, since a trivial solution is to shrink the scene until it vanishes.

Eq. 4.7 is quite robust and does not depend strongly on the initialization. We tried other initializations, e.g., setting $\mathbf{t}_A = [0, 0, 0]^\mathsf{T}$, which worked well. We have not encountered any scene in our experiments where the optimization got stuck in a poor local minimum.

### 4.2.3 Stitching

Now that we have 3D aligned depth photos, our next goal is to stitch them into a seamless panoramic mosaic. This enables removing outliers in the depth maps and also makes rendering faster by removing redundant content.

First, we compute a center of projection for the panorama by tracing the camera front vectors backwards and finding the 3D point that minimizes the distance to all of them. Then, we render all the color and depth maps from this central viewpoint into equirectangular panoramas (see supplementary document for details). The full panoramas are $8192 \times 4096$ pixels, though for each image, we only keep a crop to the tight bounding box of the pixels actually used.

As in previous work, we formulate the stitching as a discrete labeling problem, where we need to select for every pixel $p$ in the panorama a source image $\alpha_p$ from which to fetch color and depth.

#### 4.2.3.1 Data Term:

A "good" source $\alpha_p$ for the target pixel $p$ should satisfy a number of constraints, which we formulate as penalty terms.

**Depth Consensus:** If a source has the correct depth, it tends to be consistent with other views. Therefore, we count how many other views $n(p, \alpha_p)$ are at similar depth, i.e., their depth ratio is within $[0.9, 1.1]$, and define a depth consensus penalty, similar to Chapter 3:

$$E_{consensus}(p, \alpha_p) = \max\left(1 - \tfrac{n(p,\alpha_p)}{\tau_{consensus}},\ 0\right), \tag{4.8}$$

where $\tau_{consensus} = 5$ determines how many other depth maps need to agree before the penalty reaches zero and we consider the labeling to be completely reliable.

**Image Boundaries:** We prefer pixels from the image center, because there the depth maps are more reliable and there is more space for seam-hiding feathering around them. We define an image boundary penalty $E_{boundary}(p, \alpha_p)$ that is set to 1 if a source pixel is close to the boundary in its original image (i.e., within 5% of the image width), and 0 otherwise.

**Saturated Pixels:** To maximize detail in the resulting panorama, we define a term that avoids overexposed source pixels:

$$E_{saturated}(p,\alpha_p) = \begin{cases} 1 & \text{if } l(p,\alpha_p) > \tau_{saturated} \\ 0 & \text{otherwise,} \end{cases} \tag{4.9}$$

where $l(p,\alpha_p)$ is the luminance of a source pixel, and $\tau_{saturated} = 0.98$.

**Combined Objective:** By putting together the previous objectives we obtain the per-pixel data term:

$$E_{data} \;=\; E_{consensus} \;+\; \lambda_3 E_{boundary} \;+\; \lambda_4 E_{saturated}\,, \tag{4.10}$$

with the balancing coefficients $\lambda_3 = 1$, $\lambda_4 = 3$.

#### 4.2.3.2 Optimization:

Independently optimizing Eq. 4.10 for every pixel is fast, but yields noisy results since labels may change very frequently. The canonical way to achieving smoother results is to define a pairwise smoothness term that encourages fewer label changes that are placed in areas where they tend to be least visible. However, this makes the problem considerably harder and requires using slow MRF solvers. For example, in Chapter 3 we report runtimes of several minutes for solving a downscaled stitching problem.

We found that we can achieve very similar looking results faster with independent per-pixel optimization, by applying a variant of cost-volume filtering [63] which first filters the data term with a depth-guided edge aware filter:

$$E_{soft\text{-}data}(p,\alpha_p) \;= \sum_{\Delta \in W_d^\alpha} w_d^\alpha(p, p+\Delta) \cdot E_{data}(p,\alpha_p). \tag{4.11}$$

However, instead of using a single global guide, we determine unique filter weights $w_d^\alpha$ for each source image $\alpha$ using a guided filter [154], guided by the normalized *disparities* in $\alpha$. In our experiments, we use a filter footprint that spans 2.5% of the image width and set the edge-aware parameter $\varepsilon = 10^{-7}$.

In Figure 4.5 we compare our result with an MRF solution using the color and disparity smoothness terms defined in Section 3.2.4.2. While our stitch exhibits more frequent label changes the stitched color and depth mosaics are visually very similar.

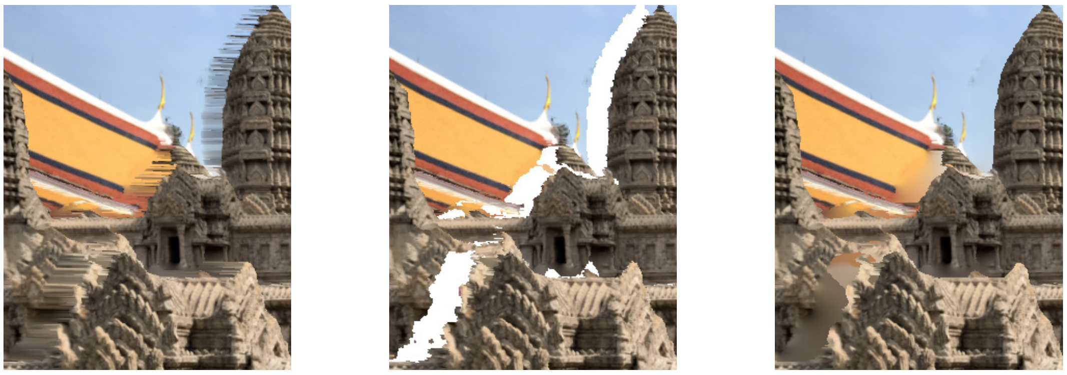

(a) Naïve mesh (b) Naïve mesh + tears (c) Multi-layer mesh

**Figure 4.6:** (a) Naïvely meshing by connecting all vertices yield stretched triangles at depth edges. (b) Tearing the mesh avoids this, but reveals holes. (c) Our multi-layer meshes extend the back-side at depth edges smoothly into the occluded region and reveal inpainted colors and depths.

#### 4.2.3.3 Color Harmonization:

Since we capture images with auto-exposure enabled, we need to align the exposures to create a seamless panorama. Following the insight from Reinhard et al. [159], we convert the images to the channel-decorrelated CIELAB color space, and then process each channel independently. We solve a linear system to compute global affine color-channel adjustments (i.e., scale and offset) for each source image, such that the adjusted color values in the overlapping regions agree as much as possible. We further reduce visible seams by feathering the label region boundaries with a wide radius of 50 pixels. In Appendix C.1 we provide more implementation details.

### 4.2.4 Multi-layer Processing

The final step of our algorithm is to convert the panorama into a triangle mesh that can be rendered on any device using standard graphics engines. Naïvely creating a triangle mesh by connecting all pixels to their 4-neighbors yields stretched triangles at strong depth edges that are revealed when the viewpoint changes (Figure 4.6a). Our solution resembles somewhat the “two-layer merging” in Chapter 3. However, an important difference is that our stitcher does not produce back-surface stitches, since the baseline is too small to reconstruct significant content in occluded regions (while we use similar camera trajectories the field of view of our camera is smaller, hence there is less overlap between images). If scenes were captured with a wider baseline and/or more wide-angle camera the two-layer stitch-and-merge algorithm mentioned above could be adapted at the expense of slower runtime.

In our algorithm, every mesh vertex corresponds to a pixel position in the panorama that is “pushed out” to a certain depth. Each vertex is connected to at most one neighbor in each of the 4 cardinal directions. Since our goal is to generate multiple layers, there can be multiple vertices at different depths for a single pixel position. We initialize the mesh by creating vertices for every panorama pixel and connecting them to their 4 neighbors.

We start the computation by detecting major depth edges in the mesh. Since the depth edges are soft and spread over multiple pixels, we apply a $9 \times 9$ median filter to turn them into step edges. Next, we tear the connection between neighboring vertices if their disparity differs by more than $5 \cdot 10^{-2}$ units. Sometimes the median filter produces small isolated “floating islands” at the middle of depth edges. We detect these using connected component analysis and merge them into either foreground or background, by replacing their depth with the median of depths just outside the floater. Figure 4.6b shows the mesh after tearing it at depth edges.

Next, we hallucinate new content in occluded parts by iteratively growing the mesh at its boundaries. In every iteration, each vertex that is missing a connection in one of the 4 cardinal directions grows in this direction and generates a new vertex at the same depth as itself. We connect new vertices with all neighboring boundary vertices whose disparity is within the threshold above. After running this procedure for a fixed number of 30 iterations, we prune the newly generated vertices by removing any but the *furthest* generated vertex at every pixel location. If the remaining generated vertex is in front of the original stitch we remove it as well. We synthesize colors for the newly generated mesh parts using diffuse inpainting. The resulting mesh smoothly extends the back-side around depth edges into the occluded regions. Instead of stretched triangles or holes, viewpoint changes now reveal smoothly inpainted color and depth content (Figure 4.6c).

In Appendix C.3 we provide some implementation details about simplification and texture atlas generation for the final mesh.

## 4.3 Results and Evaluation

We have captured and processed dozens of scenes with an iPhone 7 Plus. 25 of these are included in here, see Figure 4.7, as well as the accompanying video and the supplementary material. These scenes span a wide range of different environments (indoor and outdoor, urban and natural) and capture conditions (day and night, bright and overcast). The scenes we captured range from about 20 to 200 source images, and their horizontal field-of-view ranges from 60° to 360°.

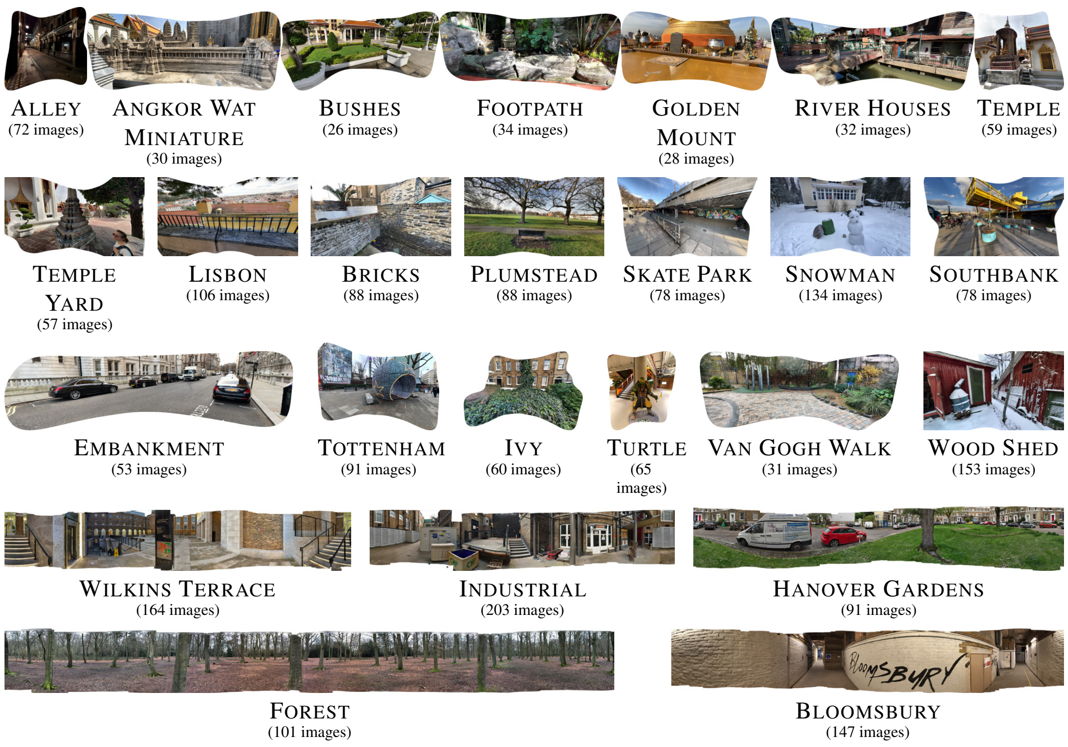


**Figure 4.7:** Datasets that we show in this chapter, video, and supplementary material. The two bottom rows show 360° panoramas.

**Table 4.1:** Breakdown of the algorithm performance per stage for the 34-image scene from Figure 4.1.

| **Stage** | **Desktop Timing** | **Laptop Timing** |
|---|---|---|
| Feature extraction and matching | 6.6s | 7.0s |
| Deformable alignment | 9.8s | 10.2s |
| Warping | 6.6s | 5.1s |
| Stitching | 2.7s | 2.6s |
| Color harmonization | 1.6s | 1.8s |
| Multi-layer computation | 1.0s | 1.1s |
| Mesh simplification | 3.1s | 3.5s |
| Texture atlas generation | 3.3s | 3.7s |
| **Total** | **34.7s** | **35.0s** |

### 4.3.1 Performance

All scenes were processed using a PC with 3.4 GHz 6-core Intel Xeon E5-2643 CPU and a NVIDIA Titan X GPU and 64 GB of memory. Our implementation mostly consists of unoptimized CPU code. The GPU is currently only (insignificantly) used in the warping stage. We ran our system also on a slower 14” Razer Blade laptop with a 3.3 GHz 4-core Intel i7-7700HQ CPU and a NVIDIA GTX 1060 GPU. Interestingly, the warping stage performs faster on the laptop, most likely because CPU computation and CPU/GPU transfers dominate

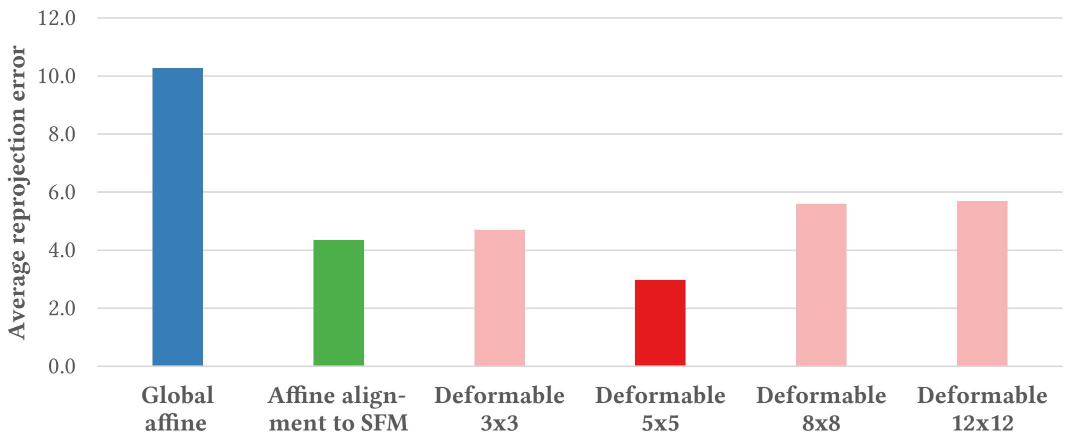


**Figure 4.8:** Average reprojection error (Eq. 4.1, *without* the robust loss function) for different alignment methods, as well as our deformable alignment with different grid sizes.

the runtime. Table 4.1 breaks down the timings for the various algorithm stages on both of these systems for an example scene. While our algorithm already runs fast, we note that there are significant further optimizations on the table. Since the deformable alignment has proven to be quite robust, we could replace the feature point detector and descriptor with faster to compute variants, e.g., FAST features and BRIEF descriptors. The alignment optimization could be sped up by implementing a custom solver, tailored to this particular problem. Our current warping algorithm is implemented in a wasteful way. Properly rewriting this GPU code would make this operation practically free. The stitching algorithm could be reimplemented on the GPU.

### 4.3.2 Alignment

Figure 4.8 shows a quantitative evaluation of our alignment algorithm. We processed the 25 scenes in Figure 4.7 using different variants of the algorithm and evaluate the average reprojection error (Eq. 4.1).

We also evaluate the effect of varying the grid size of our deformable model, which shows that the reprojection error remains flat across a wide range of settings around our choice of $5 \times 5$.

### 4.3.3 Single-image CNN Depth Maps

We experimented with other depth map sources. In particular, we were interested in using our algorithm with depth maps estimated from single images using CNNs, since this would enable using our system with regular *single*-lens cameras (even though the depth map quality produced by current algorithms is low). In Figure 4.9 we used the Monodepth algorithm [81]

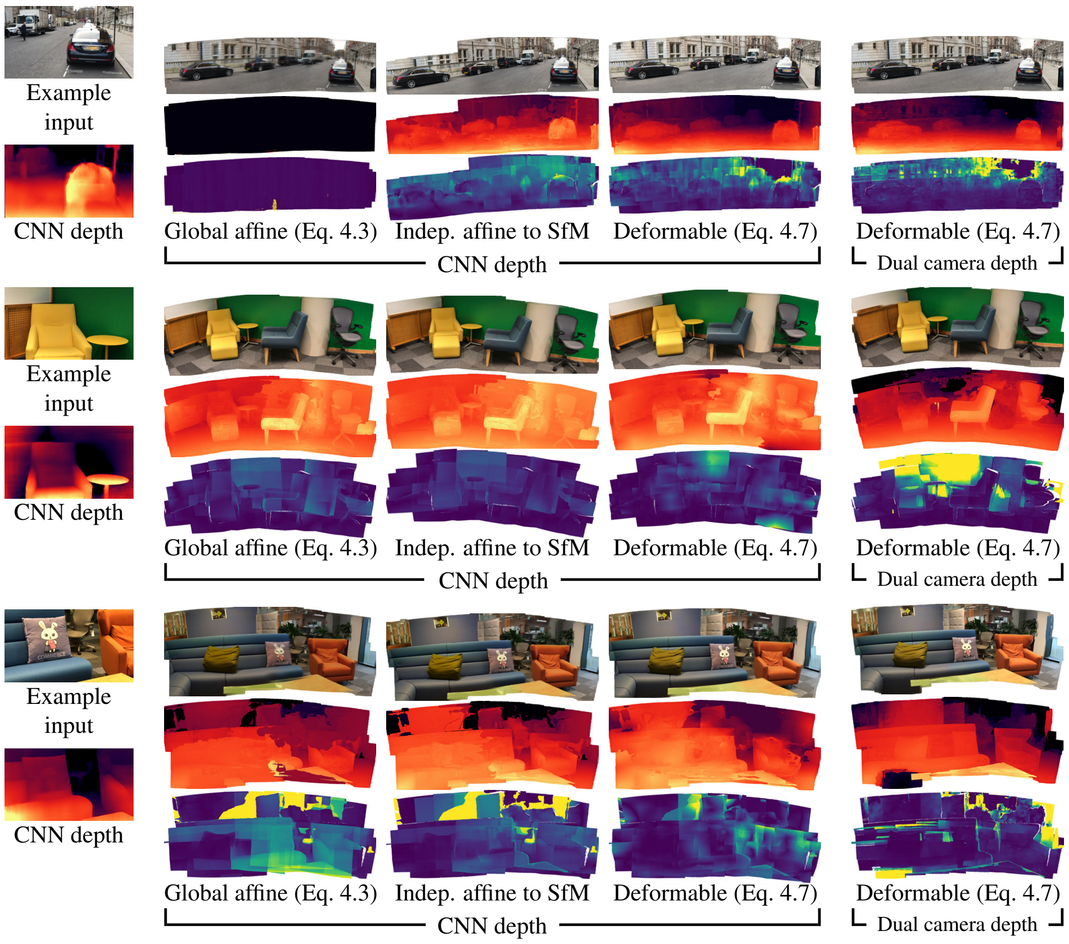


**Figure 4.9:** Applying our algorithm to single-image CNN depth maps (3 middle columns), and comparing to a result for dual camera depth maps (right column). See the supplementary material for videos of the results.

to generate single-image depth maps for three of our scenes. The original Monodepth algorithm works well for the first street scene, since it was trained on similar images. For the remaining two scenes we retrained the algorithm with explicit depth supervision using the RGBD scans in the ScanNet dataset [76]. Even though the input depth maps are considerably degraded our method was able to reconstruct surprisingly good results. To better appreciate the result quality, see the video comparisons in the supplementary material.

### 4.3.4 SfM and MVS Comparison

We were interested in how standard SfM algorithms would perform on our datasets. When processing our 25 datasets with COLMAP's SfM algorithm 7 scenes failed entirely, in 7 more not all cameras were registered, and there was 1 were all cameras registered but the reconstruction was an obvious catastrophic failure. This high failure rate underscores the

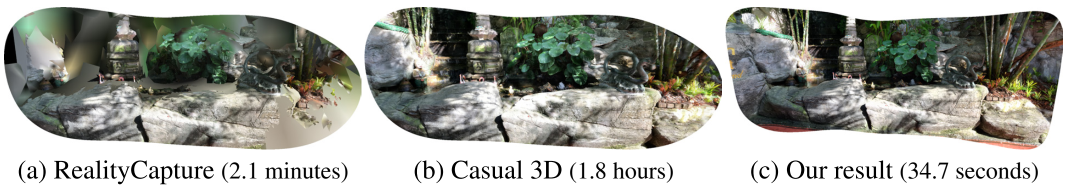

(a) RealityCapture (2.1 minutes) (b) Casual 3D (1.8 hours) (c) Our result (34.7 seconds)

**Figure 4.10:** Comparison against MVS systems: (a) Capturing Reality processes fast, but the reconstruction breaks down just a few meters away from the vantage point due to triangulation uncertainty. (b) Casual 3D produced a high quality result, but it is slow. (c) Our result has even better quality, and was computed over **200**× faster.

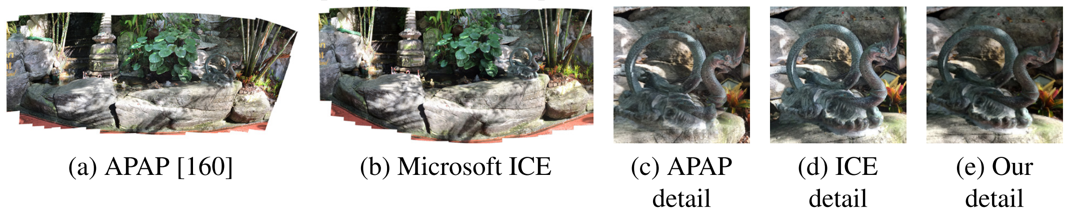

(a) APAP [160] (b) Microsoft ICE (c) APAP detail (d) ICE detail (e) Our detail

**Figure 4.11:** Comparison against monocular panoramas stitched with (a) As-Projective-As-Possible warping, and (b) Microsoft ICE. Note that these algorithms do not produce a stitched depth map. (c)-(d) show a detail crops from before mentioned algorithms. (e) corresponding detail crop from our result in Figure 4.10c.

difficulty of working with small baseline imagery.

We also compare against end-to-end MVS systems, in particular the commercial Capturing Reality system[2] and the Casual 3D photography system described in Chapter 3. Capturing Reality's reconstruction speed is impressive for a full MVS algorithm, but due to the small baseline it is only able to reconstruct foreground. Casual 3D produces results of comparable quality to ours, but at much slower speed. Figure 4.10 shows an example scene, and the supplementary material contains video comparisons of more scenes.

### 4.3.5 Parallax-aware Stitching

We compared our algorithm with two monocular stitching algorithms that handle parallax in different ways. As-projective-as-possible warping (APAP) [160] allows local deviations from otherwise globally projective warps to account for misalignment (Figure 4.11a). Note, that this does not always succeed (see detail crop in Figure 4.11c). Microsoft ICE[3] uses globally projective warps, but leverages carefully engineered seam finding and blending to hide parallax errors (Figure 4.11b and detail crop in 4.11d). Neither of these methods produce a depth panorama. While the source depth pixels could be stitched according to the label maps produced by these algorithms, this would not lead to good results, because the

[2]`https://www.capturingreality.com`
[3]`https://www.microsoft.com/en-us/research/product/computational-photography-applications/image-composite-editor/`

source depth maps are inconsistent and show the scene from different vantage points. Our algorithm resolves these inconsistencies and produces a coherent color-and-depth panorama (Figure 4.10c and detail crop in 4.11e).

### 4.3.6 Capture without Parallax

We evaluated the effect of varying the amount of parallax in the input images by capturing scenes while rotating the camera around the optical center without translating (as much as was possible), and comparing against a normal capture where we move the camera at half-arm's length. The resulting panoramas are visually very similar (see supplementary material). That said, theoretically our method should break down in the complete absence of parallax because the reprojection error in Equation 4.1 will become invariant to depth in this case. In practice, however, it is very difficult to completely avoid any parallax in the capture, and, fortunately, the natural way to capture panoramas is on an arc with a radius of about a half arm's length anyway.

### 4.3.7 Limitations

Our algorithm has a variety of limitations that lead to interesting avenues for future work.

**Capture:** The iPhone camera has a very narrow field-of-view in depth capture mode, because one of the lenses is wide-angle and the other a telephoto lens. If both lenses were wide-angle we would need to capture considerably fewer images to achieve the same amount of overlap. At the same time the baseline would increase, making the reconstruction problem easier.

**Artifacts:** Our results exhibit similar artifacts as other 3D reconstruction systems. In particular, these are floating pieces of geometry, incorrect depth in untextured regions, artifacts on dynamic objects. Compared to existing systems these problems are reduced (Figure 4.10), but they are still present. To examine these artifacts carefully we suggest watching the video comparison in the supplementary material.

**Multi-layer processing:** The hallucination of occluded pixels is rudimentary. In particular the simple back-layer extension algorithm leaves room for improvement. We plan to improve the inpainting of colors using texture synthesis.

## 4.4 Conclusions

In this chapter, we have presented a fast end-to-end algorithm for generating viewpoint-free photos suitable for seated VR from a sequence of color-and-depth images. Even though the input depth maps contain a considerable amount of low-frequency error, our novel deformable alignment optimization is able to align them precisely. This opens up the possibility to achieving a speed-up of two orders of magnitude compared to the system in Chapter 3, by replacing discrete smoothness optimization in the stitcher with independently optimizing every pixel.

We are excited about the many avenues for further improvement and research that this work opens up. Considering the performance discussion in Section 4.3.1 we believe a near-interactive implementation directly on the phone is within reach.

The results using single-image CNN depth maps are very promising. Even though the quality of these depth maps is still quite low, our alignment algorithm was able to conflate them, and stitching them using the consensus data term reduced artifacts further. Improving upon these results is an interesting direction for future research and holds the promise of bringing fast viewpoint-free photography to billions of regular cell phones with monocular cameras.

We have seen already how the availability of a fast capture and reconstruction system has changed our own behavior with respect to viewpoint-free photography. The way we capture scenes has become more opportunistic and impulsive. Almost all of the scenes in this chapter have been captured spontaneously without planning, e. g., while travelling.

This concludes our investigation into seated VR experiences. In the following chapters, we shift our focus towards the challenges with viewpoint-free photography for room-scale VR experiences.

Chapter 5

# Scalable Free-Viewpoint Image-Based Rendering

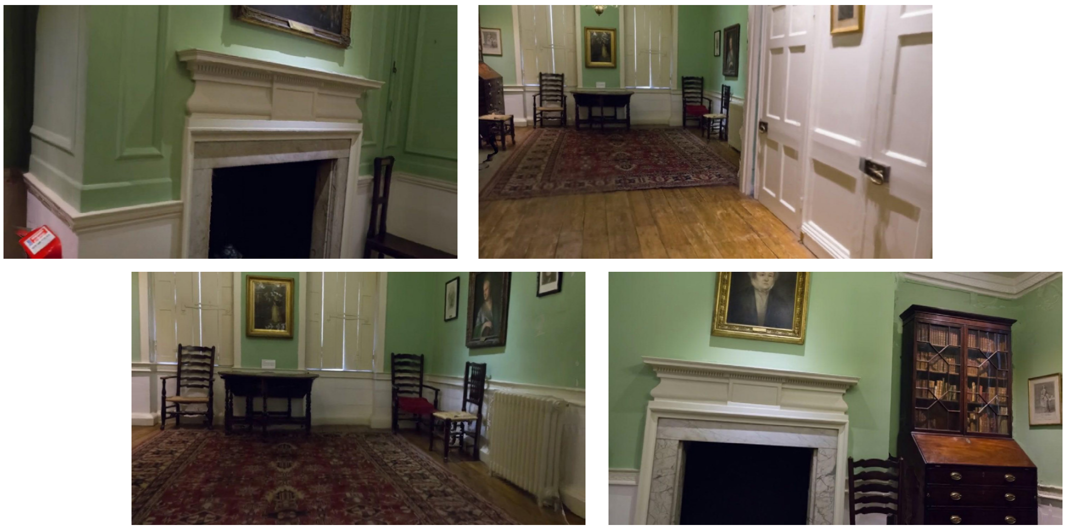


**Figure 5.1:** Images from our method rendered in 1080p at 55 Hz on an Nvidia Titan X GPU. Input is 298 high-quality photos of 'Dr Johnson's house', London. With no wheelchair access to this floor, curators were keen to have their rooms digitized.

In this chapter, we present a novel viewpoint-free photography approach for room-scale VR experiences. Compared to seated VR experiences this scenario introduces several new challenges. For example, the 3D photo representation used in the previous two chapters only captures the front-facing side of the scene and is not suitable for room-scale VR exploration, where the user is free to walk behind objects. As an alternative, we can use modern multi-view stereo (MVS) tools to obtain complete, two-sided 3D reconstructions with texture [18]. However, these reconstructions can appear blobby and seldom align with image edges. Even with perfect geometry this looks artificial, as highlights are baked in or missing completely.

The image-based rendering (IBR) approaches described in Section 2.3 achieve both realism and interactivity. However, they cannot directly be used for room-scale VR, as they generally support a very limited range of motion [23, 24], sometimes only strict interpolation between input views [14, 133, 22]. Nevertheless, a key to recent success in IBR is the use of *per-view input image information*, using representations such as custom meshes and super-pixel over-segmentation, which preserve depth boundaries even with imperfect 3D reconstructions. However, the number of images required for room-scale VR exploration is very high, making per-view geometry costly.

While MVS algorithms now provide impressive 3D reconstructions, they cannot achieve the quality required for realistic viewpoint-free photography. A first set of MVS algorithms (e. g., [17]) provides high-quality depth maps containing fine details, while a second set provides better globally consistent geometry, i. e., they estimate a smooth connected surface, even in hard-to-reconstruct (e. g., textureless) regions (e. g., [18, 161]). The geometry provided by either method still result in artifacts if used directly for viewpoint-free photography by existing IBR algorithms (e. g., Buehler et al. [127]). We develop a new per-view geometry refinement method that overcomes the shortcomings of each of the two sets of reconstruction methods. We do this by using the information available in the one set of methods as a prior to compensate for information lacking in the other. Our solution generates per-view geometry that is of sufficient quality to allow viewpoint-free photography for room-scale VR.

This chapter addresses two key challenges for room-scale viewpoint-free photography: (1) how to combine a global 3D mesh with per-view geometry, and (2) how to render high-quality novel views using this geometry, while maintaining real-time display rates.

## 5.1 Overview

Our goal is to achieve viewpoint-free photography suitable for room-scale VR experiences, using a collection of high-resolution digital color photographs as input (Figure 5.2a). Unlike traditional methods for 3D reconstruction [18, 17] and texturing [124, 123], we aim to reproduce view-dependent appearance such as highlights, and to render accurate images even in regions where a global 3D reconstruction of the scene has missing or inaccurate data.

Accurately estimating camera poses for the input photographs (Figure 5.2b) and creating a global 3D reconstruction of the scene (Figure 5.2c) is necessary for consistent rendering. However, these steps alone are not sufficient to achieve high quality with view-dependent textures (e.g., [127, 129]), since object boundaries in the photographs seldom align with the global geometry. Rather than relying on improved global geometry, we make a deliberate

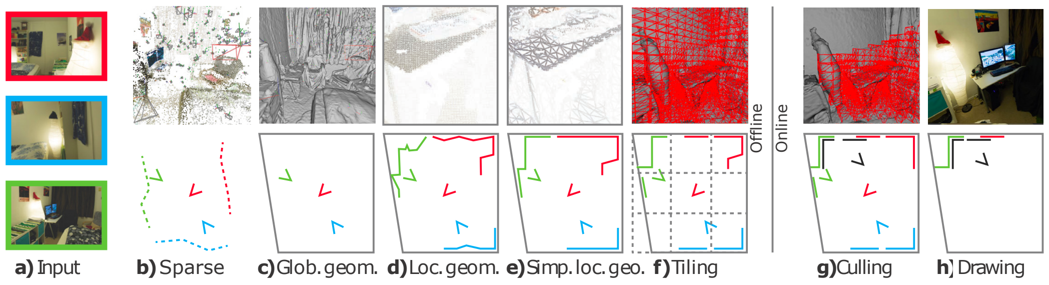


**Figure 5.2:** Algorithm overview: *(a)* The user captures photos photos by walking around the scene. *(b)* We build a sparse reconstruction of the scene using structure-from-motion software. *(c)* We use off-the-shelf software to reconstruct global geometry, which smooths away many important details, but captures the general structure of the scene. *(d)* We compute a high-quality local mesh for each input photo. *(e)* The many local meshes are simplified to reduce triangle count. *(f)* The global geometry is partitioned into tiles (in red) that organize the scene's visibility with respect to input views. *(g)* At run-time, the per-view meshes are culled, leaving only those predicted as relevant for rendering a novel view (pictured as a black camera). *(h)* The relevant per-view geometry is rendered.

trade-off: To respect object boundaries we sacrifice global agreement and create per-view geometry for each input photograph (Figure 5.2d, Section 5.2). Now, we can render novel high-quality images by forward projecting per-view geometries into the novel view, and blending them using view-dependent blend weights.

Even with high-quality geometry, hundreds of input photographs are required to faithfully reproduce the view-dependent appearance of indoor scenes during free view-point navigation, given the many occlusions and large parallax at close distances. However, iterating over all photographs when rendering novel views is very expensive, especially for forward projection where each photograph corresponds to a per-view mesh with more than a million vertices. To overcome this problem, we present a culling strategy with three elements: Simplified per-view geometry (Figure 5.2e, Section 5.2.3), a tiling data structure to only render potentially visible geometry (Figure 5.2f, Section 5.3.2), and worst-case cost bounds to avoid rendering per-view geometry likely to be discarded during view-dependent blending (Figure 5.2g-h, Section 5.3.1).

We now proceed to explain the 3D reconstruction (Section 5.2) and the rendering (Section 5.3) phases in detail.

## 5.2 3D Reconstruction

The input to our method is a set of photos calibrated with SfM [52]. From these, we can use MVS reconstruction methods to generate a global reconstruction of the scene [18], as well as per-view depth maps for each input photograph [17].

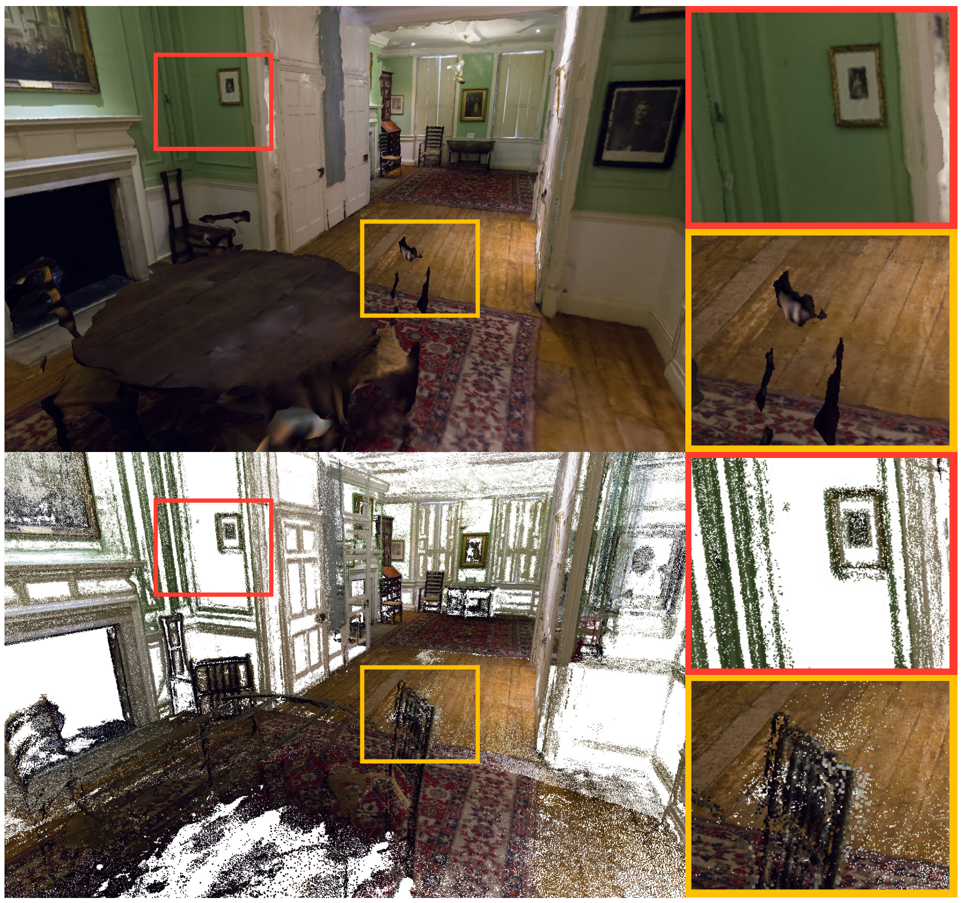


**Figure 5.3:** **Top:** Globally consistent reconstruction with RealityCapture [18]. **Bottom:** Same scene and viewpoint using COLMAP [52, 17]. We can clearly see that the feature-less wall is completely missing from the COLMAP depth maps, but a smooth estimate is provided by RealityCapture. Conversely, the details of the back of the chair are only present in the COLMAP depth maps.

In the absence of perfectly accurate global geometry, we instead seek global geometry with sufficient quality to serve as initialization for local per-view reconstruction. In general, image-based rendering methods strive to be robust against imperfect 3D geometry. However, some problems are difficult to fix with rendering alone. Scene *completeness* is one major challenge, and *accuracy* is the other. We found that global reconstructions under- or over-estimate the volume of scene elements, so they fail to align with edges in the input views. Broadly speaking, global reconstruction estimates the 3D geometry as a compromise between all input photos.

Our goal of viewpoint-free photography is less forgiving about missed occlusion boundaries, but quite compatible with slight inconsistencies in overall depth. We thus separate reconstruction into two goals: finding a global 3D structure that captures the general structure of the scene, and constructing local per-view geometry, which aligns well with image edges.

To achieve this, we combine two complementary MVS methods, one with better detail accuracy, and one with better global completeness (Section 5.2.1). Then, we run a per-pixel photoconsistency optimization to refine the occlusion edges further (Section 5.2.2). Finally, we convert the depth maps into simplified meshes amenable to efficient rendering (Section 5.2.3).

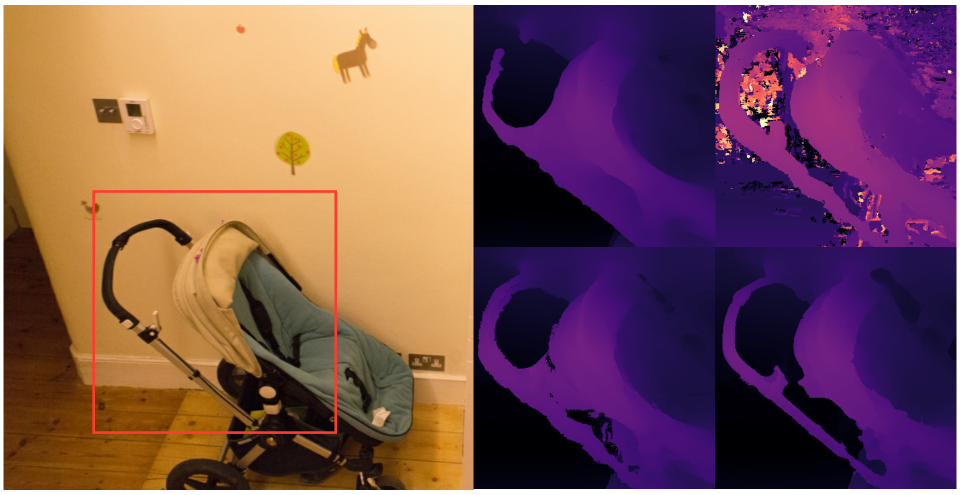


**Figure 5.4:** Merging globally and locally accurate depth maps leads to improved occlusion edge handling and artifact minimization. **Left:** Reference image **Top middle:** Globally complete RealityCapture mesh. **Top right:** Locally accurate COLMAP depth map. **Bottom middle:** Fused COLMAP and RealityCapture depth maps. **Bottom right:** Our refined depth map.

### 5.2.1 Depth map fusion

For accurate depth map reconstruction, we use COLMAP [17], which resolves small details accurately, but can break down for large textureless regions. For global meshing, we use RealityCapture [18], which is the commercial evolution of the CMPMVS method [161]. This method provides a smooth global mesh estimate, providing information in regions that are not reconstructed by other approaches (e.g, textureless content). The strengths and weaknesses of each method are clearly illustrated in Fig. 5.3.

We initialize our depth maps using COLMAP and strive to replace unreliable depths by the global geometry from RealityCapture. While there has been a lot of research on how to estimate confidence for stereo depth maps [162], we found that a simple approach which directly looks at the outputs from COLMAP works well for our purposes.

COLMAP runs in two stages: photometric and geometric. The photometric stage only optimizes photoconsistency during reconstruction. This results in outliers and noise for

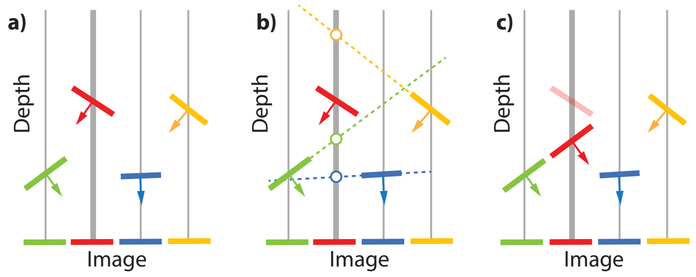


**Figure 5.5:** The spatial propagation scheme of PatchMatch optimization. Starting from pixels associated with planes (depths and normals) *(a)*, we update the plane at a pixel (in red) by first extending the planes of neighboring pixels *(b)*. We replace the plane at a pixel if any of the extended planes has a lower reconstruction energy *(c)*.

ambiguous regions such as textureless areas. In the geometric stage, a joint optimization for photoconsistency is performed, but the resulting depth maps are tested so they agree with each other in space. This correctly removes ambiguous regions, but as observed Li and Snavely [83, Supplemental Figure 1], it sometimes erodes foreground objects.

We create an initial mask for unreliable pixels in the COLMAP depth map where the photometric depth and geometric depth differs by more than 5%. Then, we smooth these masks, by applying a small guided filter [66] (guided by colors, 10 pixels wide, epsilon parameter set to 0.04). Finally, we create our fused depth maps, replacing any uncertain pixels in the COLMAP depth maps (where the mask is $> 0.5$) with the global geometry from RealityCapture.

### 5.2.2 Occlusion edge refinement

Our fused depth maps contain more details than the RealityCapture reconstruction and are more complete than the COLMAP depth maps. However, the occlusion edges are often not well aligned with image edges, since neither COLMAP nor RealityCapture were designed for this:

- RealityCapture reconstructs a global 3D mesh, which has to compromise between edge alignment in all of the input views.
- COLMAP successfully finds small structures in the scene, but often tends to produce unreliable edges, since it optimizes for photo-consistency using a patch. This is a known problem in stereo reconstruction, where patch size strikes a trade-off between edge fattening and noise [144, Fig. 12].

However, since the global geometry captures the general structure in the scene, we can use it as a strong prior to avoid noise and ambiguity, allowing us to refine occlusion edges in the depth maps using single-pixel optimization.

Our refinement uses PatchMatch optimization [163, 56] to refine pixel planes $i$ (i. e., depths and normals), searching for photoconsistent depths while preferring that they stay close to the fused depth map from Section 5.2.1. We refine the planes in a depth map $D$ by processing the pixels in raster order, one row at a time. For each pixel, we first create a set of candidate planes by extending the planes from neighboring pixels, see Figure 5.5. Then, we replace the plane at the current pixel with the candidate that has the lowest reconstruction energy:

$$E(i) = E_{\mathrm{p}}(i) + \alpha_{\mathrm{nn}} E_{\mathrm{nn}}(i), \tag{5.1}$$

which is a sum of a photoconsistency term $E_{\mathrm{p}}(i)$, and a proximity term $E_{\mathrm{nn}}(i)$ (see below). We balance the terms with $\alpha_{\mathrm{nn}} = 4 \times 10^{-4}$. Please see [163, Section 3.2] for exact details of the raster order propagation scheme.

**Photoconsistency:** We evaluate the photoconsistency $E_{\mathrm{p}}(i)$ at a pixel by reprojecting it via the candidate plane $i$ into other images, where we compare both the colors and image gradients using the formulation in [61]. We use the initial depth maps to reliably select these other images: First, we rank the other images based on overlap. Then, on a per-pixel level we use the depth maps to resolve occlusions and match against the top six images where the current candidate plane $i$ is deemed to be visible.

**Proximity:** The proximity term $E_{\mathrm{nn}}(i)$ encourages candidate planes to stay close to the fused depth map $D$ from Section 5.2.1. This is important for textureless regions, where photoconsistency does not provide any extra information. Formally,

$$E_{\mathrm{nn}}(i) = \frac{d(i,D)^2}{depth(i)^2} \tag{5.2}$$

is the depth-normalized square distance to the nearest neighbor in the original fused depth map $D$.

To achieve better edge alignment, we follow each PatchMatch iteration with a cross-bilateral weighted median filter [164] (guided by color, $\sigma_c = 0.1$) with a 4px standard deviation. We stop this optimization after two iterations (two backwards-forwards passes of PatchMatch optimization and two median filter iterations), as the change is negligible after this for all our test scenes.

As a post-process, we remove pixels at unusual depths using a geometric consistency filter (using a depth threshold $\tau_r = 1.1$, a free space threshold $\tau_f = 3$, and consensus threshold $\tau_c = 1$ — without weighted median post-processing, see Section 2.2.4.1 for details).

The resulting depth maps both preserve small features and provide reliable information in textureless regions. In Fig. 5.4, we show how our approach combines the advantages of both sources of 3D.

### 5.2.3 Representation

Instead of storing full per-pixel per-input view depth maps, we create a compressed, per-view polygonal mesh for faster drawing and reduced storage. To this end, we convert every pixel in the depth map to a 3D vertex and form a grid mesh by connecting neighboring vertices with triangles. To avoid connecting foreground and background layers at occlusion boundaries, we do not create triangles at edges where the depth difference between the neighboring pixels is large. We call those edges *splits* in the mesh. Finally, mesh simplification [165] is used to reduce the number of triangles in the mesh. We use custom simplification costs to preserve detail in foreground regions and at occlusion boundaries. As this approach was developed for another student's thesis, please refer to the description in [7, Section 4.2] for more details.

## 5.3 Rendering

We can now use the simplified per-view meshes for free-viewpoint rendering by blending many input views. To achieve high quality, including view-dependent effects such as highlights, we introduce an adaptive blending cost (Section 5.3.1). Directly rendering the per-view meshes is however prohibitively expensive, so we introduce two acceleration methods (Section 5.3.2). First, we present a spatial tiling data structure, limiting the per-view meshes used to render a given novel view. The number of tiles drawn is further reduced using a priority scheme, by deriving a bound on the blending cost for each tile. Details of our GPU rendering implementation (Section 5.3.3) conclude this section.

### 5.3.1 Blending

We form the novel view by locally blending the input images. Choosing appropriate blend weights is key to capturing view-dependent effects while minimizing ghosting artifacts. We do this by first resolving visibility of the novel view. Then, we compute a per-pixel cost for every input view. Finally, we reconstruct the novel view by adaptively blending the visible input views based on this cost.

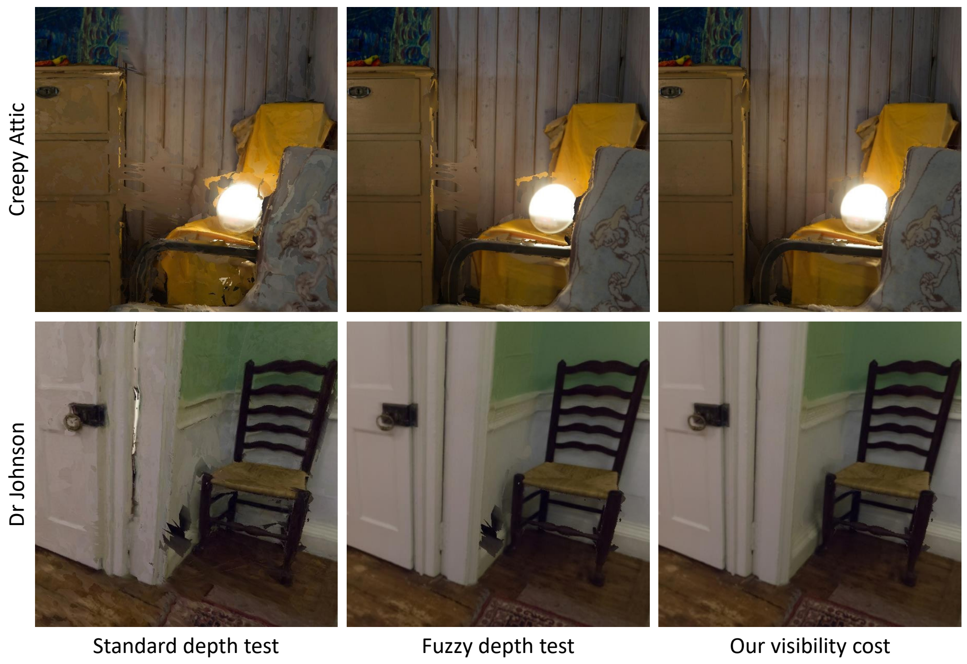


**Figure 5.6:** The impact of different depth testing strategies on image quality. **Left:** A standard depth test only preserves the front-most surfaces, and effectively disables view-dependent blending. **Middle:** A fuzzy depth used by many splatting approaches [126] approaches makes view-dependent blending possible, but cannot suppress outlier geometry. **Right:** Our visibility cost addresses both these issues.

**Resolving Visibility** As we form the final image by blending multiple per-view geometries, we cannot resolve visibility with a standard depth test that only recovers the front-most surface (see Fig. 5.6). Instead we use a visibility cost allowing us to consider all per-view geometries that represent the same surface, even if they do not perfectly align.

**Cost** We derive an IBR cost that prioritizes which input views to blend for every pixel in the novel view. This is a function $c_{\text{IBR}}(\mathbf{y}_i, \mathbf{y}_n, \mathbf{x})$ of the camera position $\mathbf{y}_i$ for the input view $i$, the camera position $\mathbf{y}_n$ for the novel view and a location $\mathbf{x}$ on the per-view geometry. Our cost is a weighted sum of three terms $c_{\text{IBR}}(\mathbf{y}_i, \mathbf{y}_n, \mathbf{x}) = \gamma_a c_\text{a} + \gamma_v c_\text{v} + (1 - \gamma_a - \gamma_v) c_\text{d}$, defined as follows:

$$c_\text{a} = \alpha(\mathbf{y}_i \to \mathbf{x} \to \mathbf{y}_n) \qquad \text{(Angle term)} \qquad (5.3)$$

$$c_\text{v} = \text{clamp}_{[0,1]}\left(\frac{d_i - d_{min}}{d_{min}}\right) \qquad \text{(Visibility term)} \qquad (5.4)$$

$$c_\text{d} = \max\left(0, 1 - \frac{||\mathbf{y}_n - \mathbf{x}||}{||\mathbf{y}_i - \mathbf{x}||}\right) \qquad \text{(Distance term)} \qquad (5.5)$$

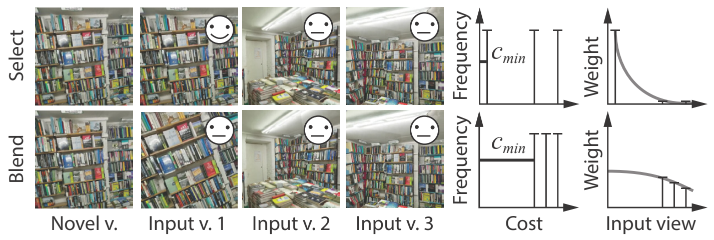


**Figure 5.7:** Adaptive bandwidth selection for two cases: One with a single good input view *(top)* and the other with many mediocre ones *(bottom)*. In the "select" case, the first view matches the novel view well and has a low IBR cost. With a low minimum cost the filter bandwidth becomes narrow, assigning most of the blend weight to the best view. In the "blend" case, all views have a high cost. Consequently, the minimum cost is high and the filter bandwidth becomes wide, blending all input views with nearly equal weights.

The *angle term* is the angle between the direction vectors from $\mathbf{x}$ towards the camera locations $\mathbf{y}_i$ and $\mathbf{y}_n$. This helps to reproduce view-dependent effects, as it prioritizes input views observing the scene from viewpoints similar to the novel view. For the *visibility term*, we first compute the depth $d_{min}$ of the closest surface for each pixel. Our depth cost reaches its minimum depth at $d_{min}$, increasing linearly with the depth $d_i$ of the current sample. To ensure that the background will not be completely discarded in the presence of floating geometry, we saturate our cost at a distance of $2d_{min}$. The *distance term* depends on the ratio of the distance between the current position $\mathbf{x}$ and the input view $\mathbf{y}_i$ resp. the novel view $\mathbf{y}_n$. This reduces blur (undersampling) in the image by penalizing input views far from the surfaces visible in the novel view. Input views closer than the novel view are not penalized, and we correct for oversampling using mip maps and anisotropic filtering. In all experiments, we set $\gamma_a = 0.45$ and $\gamma_v = 0.5$.

**Bandwidth Selection** We form the novel view by blending together the input views using blend weights determined by IBR cost. Specifically, we compute the blend weight $w_i = \exp(-c_i/\sigma)$ for an input view $i$ by applying an exponential filter kernel to the IBR cost $c_i = c_{\text{IBR}}(\mathbf{y}_i, \mathbf{y}_n, \mathbf{x})$. This implies a trade-off illustrated in Figure 5.7. We can choose a wide filter bandwidth $\sigma$ resulting in blurring (i. e., giving all views a similar weight) or a narrow bandwidth implying banding (i. e., giving only one view a very large weight). Our key observation is that no single weighting function can capture two conditions that happen in practice: At one extreme, if many similar but incorrect views are available we need to select a wide bandwidth, i. e., to blur many images. Going for a single, but imperfect view would produce a sharp, but wrong result. At the other extreme, one view is much better than the others. Here, only this input view should be selected and others should not receive any

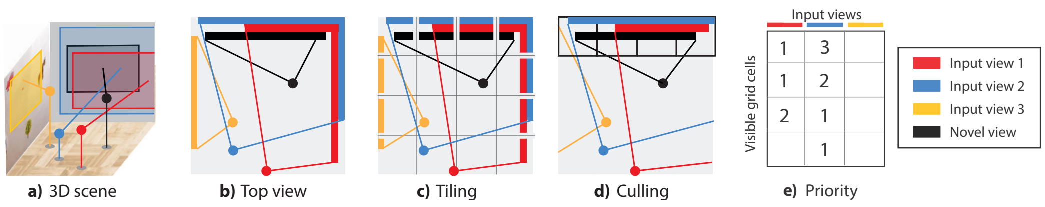


**Figure 5.8:** The tiling procedure, for a 3D scene with a single room *(a)*, acquired from three input views *(yellow, red, blue)* to rendered in the novel view (in black). With a top-down perspective *(b)*, we see that the blue and red input cameras can be seen in the black novel view, while the yellow one does not. Our method partitions the scene into a grid and splits the geometry into tiles *(c)*. Now, tiles (e. g., all yellow ones and part of red and blue) not visible in the novel view get removed *(d)*. The table in *(e)* shows the priority of each input view tile (columns) for all grid cells visible in the novel view (rows). Table entries are blank wherever the input view does not have a tile in the corresponding grid cell.

weight as blurring many views would spoil the good one.

Similarly to variable kernel density estimation, we achieve this trade-off by letting the filter bandwidth depend on the local costs of the input views. We adaptively set the filter bandwidth by scaling the filter bandwidth with the IBR cost $c_{min}$ of the best novel view. In other words $\sigma = \sigma_k c_{min}$, where $\sigma_k$ (set to 0.25 in all experiments) controls how quickly we transition from blending the input views to selecting the best one.

### 5.3.2 Tiled rendering

Even with the simplified mesh representation, drawing the per-view geometry of every input view into the novel view is slow. The key idea here is to avoid iterating over all input views when forming the novel view. We achieve this by operating on *tiles*, i. e., entire groups of input triangles and novel-view pixels. Similar ideas have been used in real-time graphics with tile-based shading [166].

The idea is shown in Figure 5.8. Consider an example of four views (black, red, blue and orange) observing a room as seen in Figure 5.8a. The black camera is the novel view, the others are input views. Figure 5.8b is a top-view of the same scene. For every pixel in the novel view (black), we need to compute IBR weights for all input views (yellow, red and blue) so we can determine which images to fetch RGB values from. With forward warping, this means that all triangles from each input view need to be drawn into the novel view.

To reduce the computational cost, we partition the scene into a regular 3D grid as shown in Figure 5.8. As a pre-process (Figure 5.8c), we associate all triangles in the input views with the grid cells they intersect. We refer to the unique pairing of a grid cell with an input view as an *input tile*, i. e., the collection of triangles from a single input view that intersect a

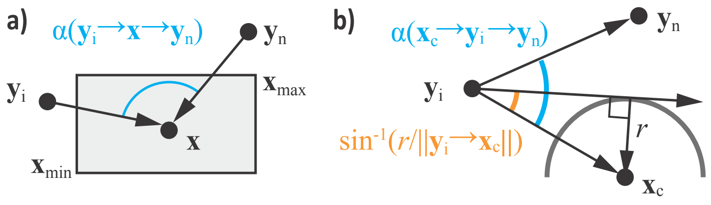

a)
$\alpha(\mathbf{y}_i \rightarrow \mathbf{x} \rightarrow \mathbf{y}_n)$
$\mathbf{y}_n$
$\mathbf{y}_i$
$\mathbf{x}_{max}$
$\mathbf{x}$
$\mathbf{x}_{min}$
b)
$\alpha(\mathbf{x}_c \rightarrow \mathbf{y}_i \rightarrow \mathbf{y}_n)$
$\mathbf{y}_n$
$\mathbf{y}_i$
$r$
$\sin^{-1}(r/\|\mathbf{y}_i \rightarrow \mathbf{x}_c\|)$
$\mathbf{x}_c$

**Figure 5.9:** We derive the upper bound on the angle $\alpha(\mathbf{y}_i \rightarrow \mathbf{x} \rightarrow \mathbf{y}_n)$, where $\mathbf{y}_i$ is an input view, $\mathbf{y}_n$ the novel view and $\mathbf{x}$ an arbitrary point in a grid cell *(a, Equation 5.7)*. We find the bound implicitly through lower bounds for the other angles in the triangle. Here, we use the bounding sphere around the grid cell (center $\mathbf{x}_c$, radius $r$) to bound $\alpha(\mathbf{x} \rightarrow \mathbf{y}_i \rightarrow \mathbf{y}_n)$ (b, *Equation 5.8)*.

grid cell. At run-time, we save effort by only rendering input tiles visible to the novel-view (tiling and culling). In Figure 5.8d, we see that this allows us to ignore the yellow view altogether as well as parts of the red and blue views.

However, because of our view-dependent blending, only a few visible input tiles will actually contribute to the final image. We therefore strive to render a small subset of the input tiles that ensures a low IBR cost for all pixels in the novel view. To achieve this, we sort the input tiles according to their worst-case IBR cost and draw only the best ones in each visible grid cell. The exact number varies between grid cells, as we stop drawing input tiles when we determine that a cell has been sufficiently covered. Predicting the worst-case IBR cost is part of our contribution, which focuses the rendering effort to the input tiles that actually affect the final image.

#### 5.3.2.1 Tiling and Culling

We store the input tiles in a 3D grid with a resolution of $32 \times 32 \times 32$ that covers the scene. During pre-processing, we create the input tiles by conservatively voxelizing the per-view geometry into the grid. As a consequence, each triangle in the input geometry may belong to multiple input tiles.

At run-time, we need to find the grid cells visible to the novel view. This is done in two passes. First, we draw the coarse global geometry into the framebuffer of the novel view. Second, we process all framebuffer pixels in parallel and mark all grid cells they intersect as visible using an atomic operation. To account for the mismatch between the coarse global geometry and the per-view input geometry, we perform this intersection conservatively by inflating each pixel into a box which matches the size of the voxels in the scene.

#### 5.3.2.2 Priorization

To select which input tiles to render, we sort them within each visible grid cell according to their worst-case IBR cost, i.e., an upper bound for $c_{\mathrm{IBR}}(\mathbf{y}_i, \mathbf{y}_n, \mathbf{x})$ between the input view $\mathbf{y}_i$ and the novel view $\mathbf{y}_n$ where $\mathbf{x}$ is allowed to be anywhere in the grid cell. Concretely, $\mathbf{x}$ is inside an axis-aligned bounding box $(\mathbf{x}_{\mathrm{min}}, \mathbf{x}_{\mathrm{max}})$, see Figure 5.9a. Using separately computed upper bounds for the angle term $c_{\mathrm{a}} \leq c_{\mathrm{a}}^{max}$, the depth term $c_{\mathrm{v}} \leq c^{max}\mathrm{v}$, and the distance term $c_{\mathrm{d}} \leq c_{\mathrm{d}}^{max}$, we form the upper bound

$$c_{\mathrm{IBR}}(\mathbf{y}_i, \mathbf{y}_n, \mathbf{x}) \leq \gamma_a c_a^{\mathrm{max}} + \gamma_v c_v^{\mathrm{max}} + (1 - \gamma_a - \gamma_v) c_d^{\mathrm{max}} \tag{5.6}$$

**Angle term:** To find an upper bound for $c_{\mathrm{a}} = \alpha(\mathbf{y}_i \to \mathbf{x} \to \mathbf{y}_n)$, we exploit that the angles in a triangle sum to $\pi$ radians. Consider the triangle between $\mathbf{x}, \mathbf{y}_i$ and $\mathbf{y}_n$. We find lower bounds $\alpha_{\mathrm{min}}(\mathbf{y}_{\mathrm{i}})$ and $\alpha_{\mathrm{min}}(\mathbf{y}_{\mathrm{n}})$ for the other two angles in the triangle and form the upper bound:

$$c_{\mathrm{a}}^{max} = \pi - \alpha_{\mathrm{min}}(\mathbf{y}_{\mathrm{i}}) - \alpha_{\mathrm{min}}(\mathbf{y}_{\mathrm{n}}). \tag{5.7}$$

Figure 5.9b shows how we compute the lower bound $\alpha_{min}(\mathbf{y}_i)$ for one of the other angles $\alpha(\mathbf{x} \to \mathbf{y}_i \to \mathbf{y}_n)$. First, we convert the bounding box $(\mathbf{x}_{\mathrm{min}}, \mathbf{x}_{\mathrm{max}})$ to its encompassing bounding sphere centered at $\mathbf{x}_c$ with the radius $\mathbf{r}$. Now,

$$\alpha_{\mathrm{min}}(\mathbf{y}_{\mathrm{i}}) = \alpha(\mathbf{x}_{\mathrm{c}} \to \mathbf{y}_{\mathrm{i}} \to \mathbf{y}_{\mathrm{n}}) - \sin^{-1}\left(\frac{r}{||\mathbf{y}_{\mathrm{i}} - \mathbf{x}_{\mathrm{c}}||}\right). \tag{5.8}$$

**Visibility term:** We use the trivial upper bound $c_{\mathrm{v}}^{max} = 1$, since we've already culled the voxels based on visibility.

**Distance term:** We derive an upper bound for the distance term $c_{\mathrm{d}}$ based on the maximum distance $d_{\mathrm{max}}(\mathbf{y})$ and minimum distance $d_{\mathrm{min}}(\mathbf{y})$ between a point $\mathbf{y}$ and a bounding box $(\mathbf{x}_{\mathrm{min}}, \mathbf{x}_{\mathrm{max}})$. Using $d_{\mathrm{min}}(\mathbf{y}_n)$ as lower bound for the numerator and $d_{\mathrm{max}}(\mathbf{y}_i)$ as an upper bound for the denominator, we see that

$$c_{\mathrm{d}}^{max} = \max\left(0, 1 - \frac{d_{\mathrm{min}}(\mathbf{y}_{\mathrm{n}})}{d_{\mathrm{max}}(\mathbf{y}_{\mathrm{i}})}\right). \tag{5.9}$$

**Drawing** Once all input tiles have been sorted, we determine how many are to be rendered from each grid cell. Ideally, we would conserve bandwidth by rendering as few input tiles as possible. However, this may result in holes in the novel view; the triangles in an input tile are not guaranteed to cover a grid cell, e. g., due to splits (Sec. 5.2.3) or image boundaries. We refer to an input tile without any of these discontinuities as *complete*.

For each grid cell, we render the sorted input tiles until three complete tiles or a maximum of 12 input tiles have been rendered. We found 12 to be a good trade-off between completeness and compactness. In our experiments this typically reduces the number of drawn primitives to fewer than 10 % of the original per-view meshes.

### 5.3.3 Implementation

Four geometry passes are required to render a novel view. Deferred shading cannot be used to avoid multiple passes, as the color of every output pixel is determined by blending several input primitives.

As described in Section 5.3.2, the first pass renders the global geometry to find the visible grid cells. These cells are downloaded to the CPU, which prioritizes the input tiles to render in the remaining three passes. We store the geometry associated with all input tiles in a single large vertex buffer object, allowing us to perform the other passes using a single `glDrawIndirect` OpenGL instruction.

The final three passes implement the blending algorithm described in Section 5.3.1. First, we recover the front-most surface with a depth pre-pass, which is required for the visibility term $c_v$. The second pass finds the minimum IBR cost $c_{min}$ of all input views that project onto each pixel. This is required for the per-pixel filter bandwidth $\sigma$ explained in Section 5.3.1. Finally, the last pass forms the image, blending together the input views using the bandwidths computed in the previous pass.

## 5.4 Experiments

We compare our approach to existing baseline algorithms, using published code wherever possible. Most existing rendering systems were meant for outside-in scenes, so results are presented on our own data. Here and in the supplemental material, we provide qualitative comparisons of our rendering phases, and quantitative results for speed tests.

**Data Sets** We captured scenes with a digital camera (Sony NEX-C3 at $1228 \times 816$ or Canon EOS 550D at $1296 \times 864$), taking 150-300 RAW photos in each scene. In a pre-processing

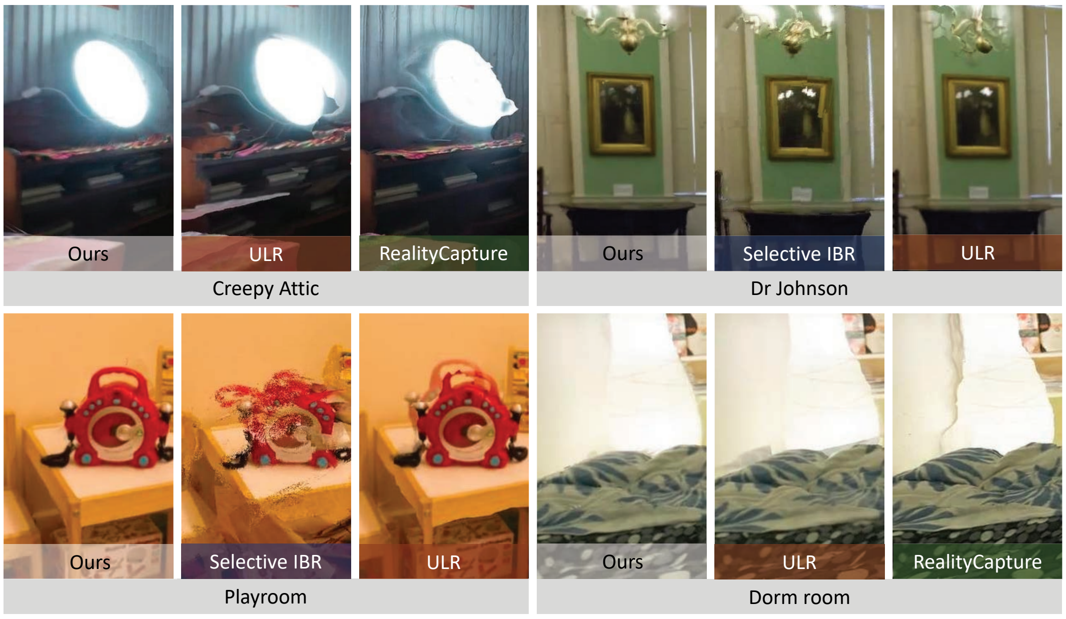


**Figure 5.10:** Four example scenes, comparing our system with ULR [127], Selective IBR et al. [132], and the global mesh from RealityCapture [18].

step, we color-harmonized the images using Adobe Lightroom. In Figure 5.1, 5.11 5.10 and the supplemental material we show results in the following scenes:

CREEPY ATTIC: A small ($5 \times 4 \times 4$ m$^3$) attic in an old building containing textiles and artwork. There is a prominent object (doll on a chair) in the middle of the room, which clearly shows the need for per-input view local geometry since under- or overestimated global geometry results in clear IBR artifacts.

DORM ROOM: A small bedroom ($4 \times 4 \times 5$ m$^3$) in a student dorm with textureless walls.

DR JOHNSON: A preserved house from the 17th century. This is a large scene ($16 \times 6 \times 5$ m$^3$) exhibiting textureless walls, glossy tabletops and paintings behind mirror-reflective glass.

PLAYROOM: A medium-sized ($6 \times 6 \times 5$ m$^3$) livingroom cluttered with toys for young children. Aside from large, textureless surfaces this scene contains many small geometric details that cannot easily be captured using geometry reconstruction alone.

In supplemental material, we also show comparisons in a collection of standard IBR scenes from [23], as well as some of our own outdoor scenes.

**Rendering Comparison** In the supplemental material and Fig. 5.10 we compare our algorithm to the global mesh reconstructed by the state-of-the-art commercial multi-view stereo tool RealityCapture [18], Selective IBR [132], and also to ULR [127] with improved

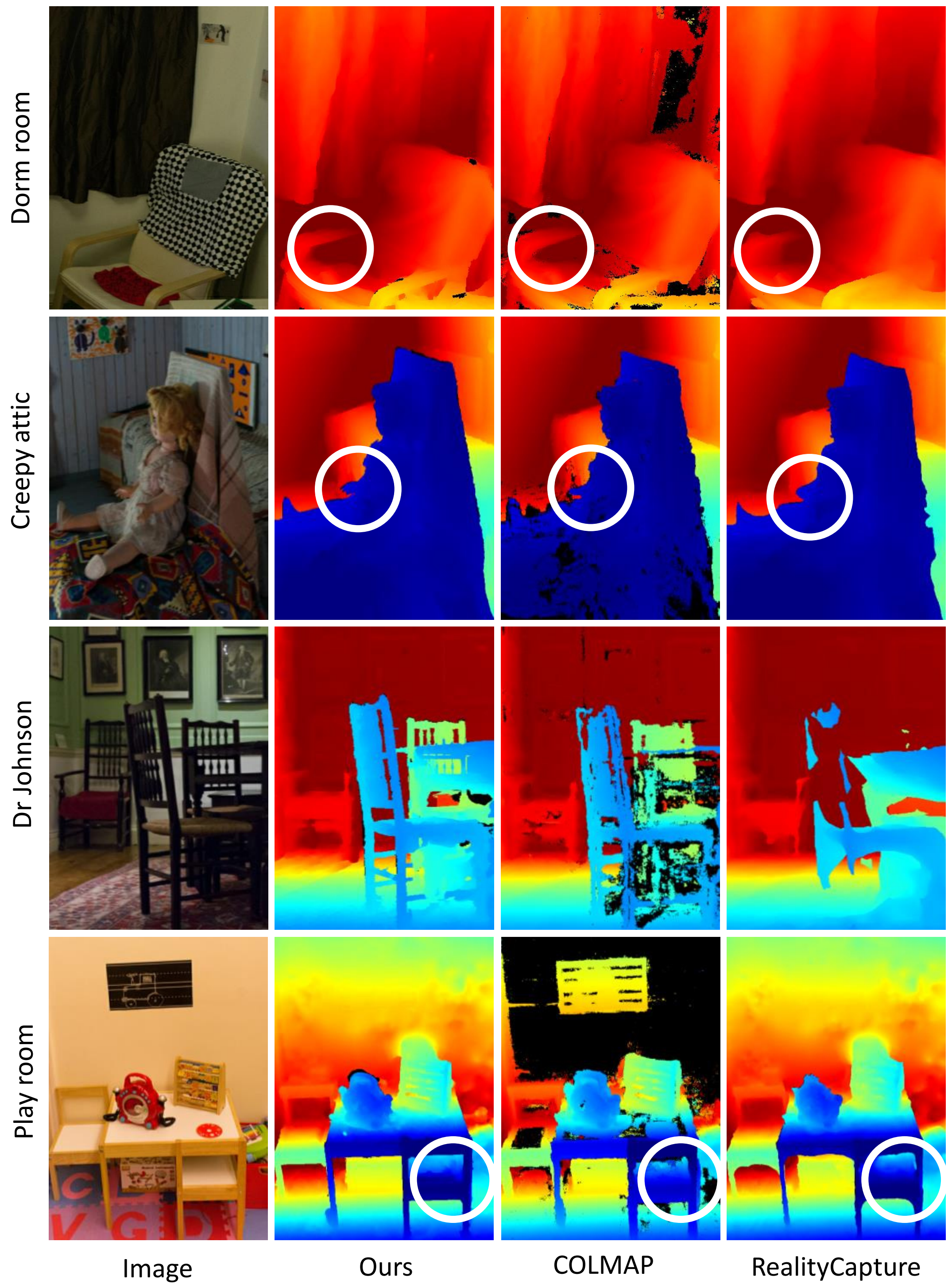


**Figure 5.11:** Different geometry reconstructions. In each scene we compare: The input photograph; Our local-global reconstruction (Section 5.2); Global reconstruction [18]; COLMAP [17]; Our local-global approach combines the best of both; complete reconstructions that align well with image edges (see the white circles).

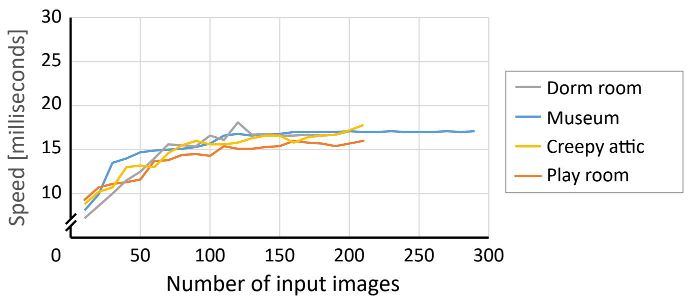


**Figure 5.12:** Performance scaling at 1080p with Nvidia GTX Titan X. We fix the camera position in each scene and measure performance of our tiled rendering as the number of input images increases.

visibility akin to floating textures [129].

Compared to [132], our mesh-based approach tends to preserve complex shape boundaries better since super-pixels are warped independently in their method. This can be seen in PLAYROOM, where the toy radio gets broken up, and also in DR JOHNSON, where the painting on the wall has visible seams.

Compared to the global mesh from RealityCapture, our geometry refinement approach helps preserve the shape of hard-to-reconstruct objects better. This can for example be seen on the lamps in both CREEPYATTIC and DORM ROOM.

Compared to ULR, our refined geometry results in significantly fewer foreground bleeding artifacts (e. g., CREEPYATTIC, PLAYROOM and DORM ROOM) and also preserves the shape of hard-to-capture objects like the chandelier in DRJOHNSON.

**Geometry Comparison** In Figure 5.11 we compare the geometry reconstructions from different components of our system. We see that the global reconstruction from RealityCapture [18] is complete but tends to over- or underestimate the size of foreground objects. We also include COLMAP [17], which preserves details better than the global reconstruction, but is often incomplete (shown as black) as textureless regions are difficult to reconstruct using multi-view stereo. The geometry produced by our local per-view refinement (Section 5.2) is complete and aligns well with image edges. Note how it corrects occlusion boundaries grossly misestimated in the global reconstructions (circled in white): The armrest in DORM ROOM; the hand of the doll in CREEPY ATTIC; the eroded chair in DR JOHNSON and the rounded corners on the chair in PLAY ROOM. Correctly reconstructing these boundaries is important for IBR, as getting them wrong causes foreground color to be displayed on background geometry or vice-versa.

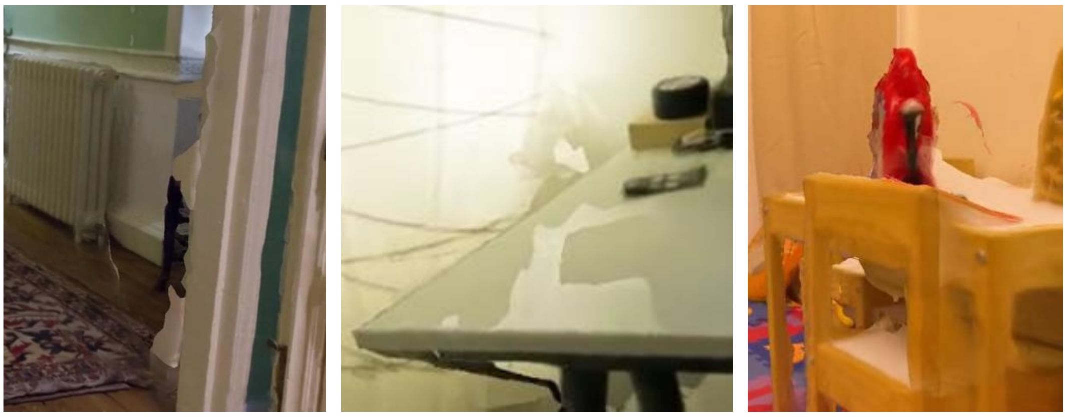


**Figure 5.13:** Typical artifacts with our approach. **Left-to-right:** Incorrect geometry at occlusion boundaries (foreground fattening), broken highlights (color harmonization), foreground bleeding into background.

**Performance** In these experiments we use the rendering system presented here, but slightly different geometry. This may impact the overall performance per scene, but has a minimal effect on the relative measurements we emphasize here. We use the geometry from [6], which uses depth-sensor data for global geometry and a slightly different refinement approach.

Table 5.1 shows show statistics of our scenes and rendering. We measured the average frame time at 1080p in each scene as the virtual camera moves along a predetermined path. Without tiling, rendering takes 35-60 ms on our high-end machine (Desktop PC; Nvidia GTX Titan X), and 220-388 ms on the low-end machine (Laptop; Nvidia GTX 660M). Tiling provides a speedup of $2\times$ to $4\times$ depending on the scene and hardware. Interestingly, tiling reduces the number of rendered triangles more drastically than frame time. This is explained by the overhead from the tiling process itself, particularly from prioritization; the only CPU component of our rendering algorithm. Figure 5.12 shows how the performance of our tiled algorithm scales with the number of input images, demonstrating a sub-linear trend where the frame time plateaus after 150 images.

## 5.5 Conclusion and Future Work

We have presented a novel approach to viewpoint-free photography for room-scale VR experiences which achieves several of the goals in Section 1.1: it works with an unstructured collection of photos as input (goal 1 - ease of capture), supports a large range of motion (goal 2), and maintains real-time display rates — even for large scenes (goal 4).

However, while this method preserves occlusion boundaries and view-dependent effects, it struggles to consistently achieve high quality results (goal 3). See Figure 5.13 for a few examples. In the following chapter we address this problem, using a convolutional neural network to alleviate these artifacts during the blending stage.

**Limitations** Our approach is ultimately limited by the quality of the initial capture; evidently SfM needs to succeed for all the input images for our method to work. Handling much larger scenes with thousands of input photographs is currently a challenge. A clever disk and main memory caching scheme is required. Finally, our tiling method is designed for relatively wide-baseline data sets. For light-field like densities, the CPU overhead for treating the tiling structure could become a bottleneck.

**Table 5.1:** Evaluation of our tiled rendering in four scenes with different levels of complexity. In each scene, we record the number of primitives needed when using our tiled method, and contrast it to a brute-force method rendering all local meshes. We render at 1080p and measure frame time in milliseconds, both on a high-end Machine A (Desktop with a Nvidia GTX Titan X) and a low-end Machine B (Laptop with a Nvidia GTX 660M).

| | Input | | Tris/frame | |
|---|---|---|---|---|
| Scene | RGB | Depth | Tiling | No tiling |
| CREEPY ATTIC | 223 | 373 | 0.32 M | 6.70 M |
| DORM ROOM | 201 | 312 | 0.35 M | 7.90 M |
| PLAY ROOM | 231 | 409 | 0.40 M | 8.40 M |
| DR JOHNSON | 298 | 450 | 0.60 M | 16.8 M |

| | Machine A | | | Machine B | | |
|---|---|---|---|---|---|---|
| | Speed-up | Tiling | No tiling | Speed-up | Tiling | No tiling |
| CREEPY ATTIC | 3.3× | 15.1 ms | 35.3 ms | 2.3× | 66.60 ms | 219.6 ms |
| DORM ROOM | 4.1× | 15.4 ms | 45.9 ms | 3.0× | 69.80 ms | 289.7 ms |
| PLAY ROOM | 3.3× | 15.4 ms | 40.8 ms | 2.7× | 81.70 ms | 269.7 ms |
| DR JOHNSON | 3.3× | 18.2 ms | 60.5 ms | 3.3× | 119.4 ms | 388.2 ms |

Chapter 6

# Deep Blending for Free-Viewpoint Image-Based Rendering

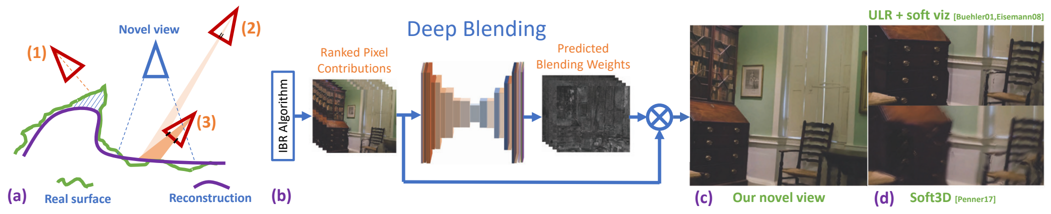


**Figure 6.1:** Image-based rendering *blends* contributions from different input images to synthesize a novel view. **(a)** This blending operation is complex for a variety of reasons. For example, incorrect visibility (e. g., for the non-reconstructed green surface in input view (1)) may result in wrong projections of an image into the novel view. Blending must also account for, e. g., differences in projected resolution ((2) and (3)). Previous methods have used hand-crafted heuristics to overcome these issues. **(b)** Our method generates a set of ranked contributions (mosaics) from the input images and uses *predicted blending weights* from a CNN to perform *deep blending* and synthesize **(c)** novel views. Our solution significantly reduces visible artifacts compared to **(d)** previous methods.

In the previous chapter, we presented an approach for viewpoint-free photography suitable for room-scale VR experiences. However, like many other Image-Based Rendering (IBR) methods [132, 24], this approach still suffers from many visible artifacts, especially when moving far from the input photos. In general, novel views are synthesized in IBR by combining warped pixels from input photos; output quality depends on the computation of visibility in the presence of *inaccurate geometry* and on the *blending* method.

With only few exceptions (e. g., [167]), previous solutions use heuristic blending to handle geometric inaccuracies, and to correct image seams and ghosting due to view-specific differences in the combined images. Blending needs to correct for artifacts due to incorrect *occlusion edges*, *visible seams* due to *texture stretch/misalignment*, and *lack of* *color harmonization*, as well as view-dependent effects from highlights, different exposures, and unsuitable camera selection. These complex, often contradictory requirements have led prior work to develop case-specific, hand-crafted heuristics that always fail for some configurations.

In this chapter, we build on the geometry refinement and rendering contributions from Chapter 5 with the following contribution: A deep convolutional neural network solution to the blending problem. Our main insight is that a data-driven solution is currently a better strategy to effectively satisfy these challenging requirements. We introduce a *deep blending* algorithm, leveraging convolutional neural networks (CNNs) that learn *blending weights* that will most reasonably approximate real imagery for novel view synthesis (Figure 6.1).

Our solution provides realistic image-based rendering across the variety of scenes attempted by the IBR community so far, and it gracefully degrades in quality when the geometric reconstruction fails completely or exposure differences are too pronounced. For a majority of the scenes tested – both outdoors and indoors – our method achieves excellent quality for free-viewpoint navigation, while also being capable of achieving interactive frame-rates.

## 6.1 Overview

Deep blending for IBR poses several challenges. We first need to provide the network with the best possible geometry to reduce the amount of artifact correction required. We then need to determine the CNN architecture and adapt previous rendering algorithms to maintain interactive free-viewpoint navigation, while tightly integrating with learning. We also need to generate sufficient training data to allow the network to learn good blending weights. Finally, we need to define a loss function that minimizes rendering artifacts. We next present an overview of how we solve each of these challenges, using the pipeline shown in Figure 6.2.

Our input is a set of input photographs of a scene. We first calibrate the cameras using structure from motion (SfM) [52] and then use the approach from Chapter 5 to create high quality per-view meshes that respect occlusion edges as much as possible.

Integrating a Neural Network into an interactive IBR system is challenging. To allow a per-frame interactive rendering loop that includes a CNN evaluation, we choose a U-net [168] architecture, and generate a fixed set of inputs to the CNN. For rendering, we build on Chapter 5, which at each output pixel selects a variable number of input photos to blend into a final image (Figure 6.2). In our rendering loop, we use the blend weights to rank these

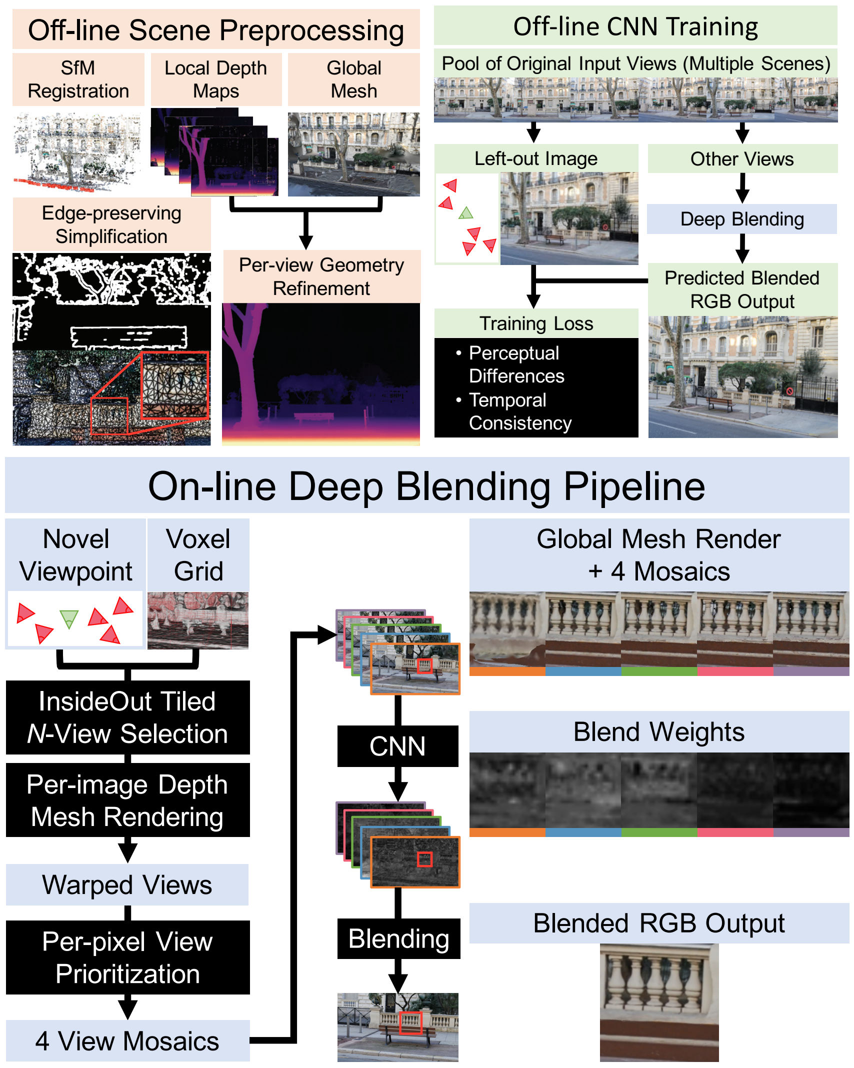

Off-line Scene Preprocessing
SfM Registration
Local Depth Maps
Global Mesh
Edge-preserving Simplification
Per-view Geometry Refinement
Off-line CNN Training
Pool of Original Input Views (Multiple Scenes)
Left-out Image
Other Views
Deep Blending
Predicted Blended RGB Output
Training Loss
• Perceptual Differences
• Temporal Consistency
On-line Deep Blending Pipeline
Novel Viewpoint
Voxel Grid
InsideOut Tiled N-View Selection
Per-image Depth Mesh Rendering
Warped Views
Per-pixel View Prioritization
4 View Mosaics
CNN
Blending
Global Mesh Render + 4 Mosaics
Blend Weights
Blended RGB Output

**Figure 6.2:** Overview of our approach. **Top left:** As described in Chapter 5, scene preprocessing entails constructing 1) 6-DoF image positions using SfM, 2) per-image depth maps using MVS, 3) a global mesh, 4) mesh-refined depth maps, and 5) simplified per-view meshes respecting occlusion edges. **Top right:** Training uses a perceptual loss to compare our pipeline's output (bottom) for a known viewpoint to the real image; a temporal consistency term penalizes differences after viewpoint perturbation. **Bottom:** Deep Blending outputs color images for novel scene viewpoints. At each output pixel, we use the blend weights from Chapter 5 to rank pixels in the dataset images. Our network takes 4 color *mosaics* of the top samples, plus a global mesh rendering, and outputs per-pixel blend weights. The weighted sum of inputs forms a new color image.

per-pixel selections to generate a fixed number of *mosaics* that are blended into the novel view. Each pixel of the first mosaic contains the color value of the best selected pixel, the second mosaic contains the second best, and so on.

Training data for our supervised learning of the CNN weights is non-traditional: the same photo serves, at times, as one of the *inputs* to the mosaic-building step, or it is held-out so it can serve as the ground truth *output* that the network tries to reconstruct from mosaics of other input photos. We generate a large dataset of input images through round-robin use of this hold-out strategy, and through data augmentation. The test results we show are distinct samples (i.e., novel viewpoints) never seen in training. Finally, to achieve good visual quality, to overcome alignment issues and to reduce flickering, we use a perceptually-motivated loss [169] and introduce a temporal coherence term. Our experiments show that the network training can be run on a general set of images from training scenes, and that performance is stable even when no images for a given testing scene were seen during training.

Our approach allows interactive rendering for many of our scenes, and provides close to photorealistic free-viewpoint navigation, even far from the input cameras. Compared to previous work, our method significantly reduces many glaring artifacts.

## 6.2 Learning to Blend

The blending step of IBR needs to compensate for geometric and visibility errors, and for view- and image-dependent effects. Correctly handling visibility for IBR in the presence of inaccurate geometry is a very hard problem that has plagued the field from the outset. Various hand-crafted heuristics include soft visibility [128, 170, 129, 6], the "prefer background" heuristic [23], and the discretized soft visibility of the Soft3D approach [24]. Similarly, final blending weights are typically heuristics, which try to address the different sources of artifacts. These include the angle and distance terms for ULR [127], the more sophisticated texture-stretch heuristics of Hyperlapse [171] – requiring an offline graphcut/Poisson blending step – or the adaptive bandwidth of from Chapter 5. Despite attempts at a more rigorous Bayesian formulation [172], reliably achieving high-quality blending effectively remains elusive.

Deep learning has demonstrated the ability to perform very complex image transformation tasks and is thus an ideal candidate for solving the IBR blending problem. We train a convolutional neural network (CNN) to generate *blending weights* that are then used to combine warped (or reprojected) contributions from different input images. Our goal is to evaluate the feed-forward CNN in an interactive renderer, imposing strict constraints on

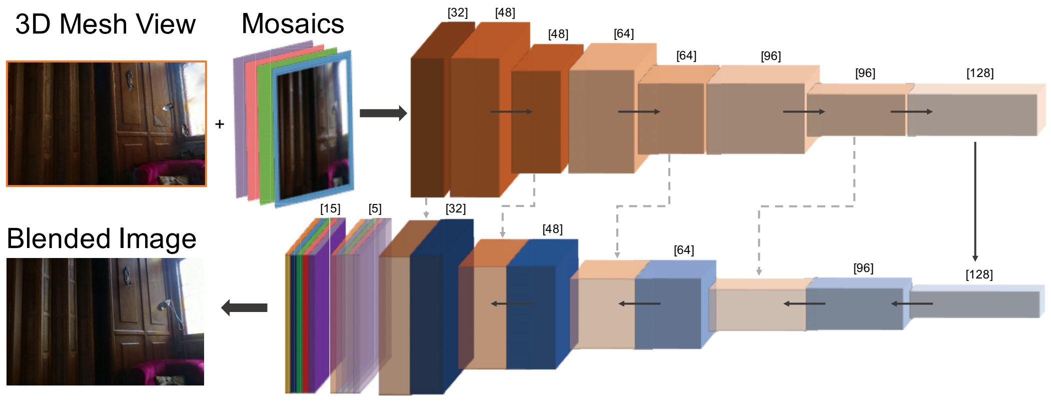


**Figure 6.3:** Deep Blending network architecture. An RGB view of the global mesh is concatenated channel-wise with 4 RGB mosaic images and fed into a fully convolutional U-net-type architecture. Each successive block is generated via a $3 \times 3$ convolution followed by a ReLU activation; the number of channels for each block is shown in brackets. Dotted lines represent "skip connections," where features in the down-convolution portion are concatenated to features in the up-convolution portion. Decreases in spatial resolution are made by convolving with a stride of two, and increases use nearest-neighbor upsampling before the convolution is applied. A softmax activation is applied to obtain per-pixel blend weights. The novel image is obtained by taking the per-pixel weighted sum of the 5 network inputs.

network architecture, input layers, and the renderer. Generating sufficient training data to find the weights is also challenging. Finally, care must be taken when defining the training loss: we want to produce images that are as realistic as possible, avoiding temporal artifacts such as flickering.

### 6.2.1 Network Architecture and Rendering Algorithm

Our goal of tight integration in the rendering loop imposes strict requirements in terms of network architecture, and precludes involved solutions such as Recurrent Neural Networks [173]. In addition, we need a CNN that is capable of handling image transformations in a multi-scale fashion. For these reasons, we choose a small U-net architecture [168] with skip connections, which has been demonstrated to work well on image transformation tasks [174].

The network architecture is shown in Figure 6.3. This architecture has a receptive field of 63 pixels at the bottleneck, in theory giving it the power to correct for artifacts that are at most this size.

This network architecture requires a fixed number of inputs, and we choose to provide it with five images, one global mesh render and 4 source-image mosaics, that need to be blended. We use the rendering algorithm in Chapter 5, to quickly find, at each output pixel, pixels in multiple input images that contribute to the novel view.

Since we have chosen to fix the number of inputs to our CNN, we need to modify the rendering algorithm to "pack" this information into four separate layers that will then be fed to the network for blending. We do this by creating four *mosaics* formed by re-ranking the set of sampled source pixels on a per-output-pixel basis. We rank source pixels using the IBR cost described in Section 5.3.1. For each pixel, we store the four most highly ranked contributions in the four mosaics. In the case of missing geometry, we fill holes in the input mosaics with a rendered view of the textured global mesh. These four mosaics, together with a view of the global mesh, form the input layers to our network. This is illustrated in Figure 6.4.

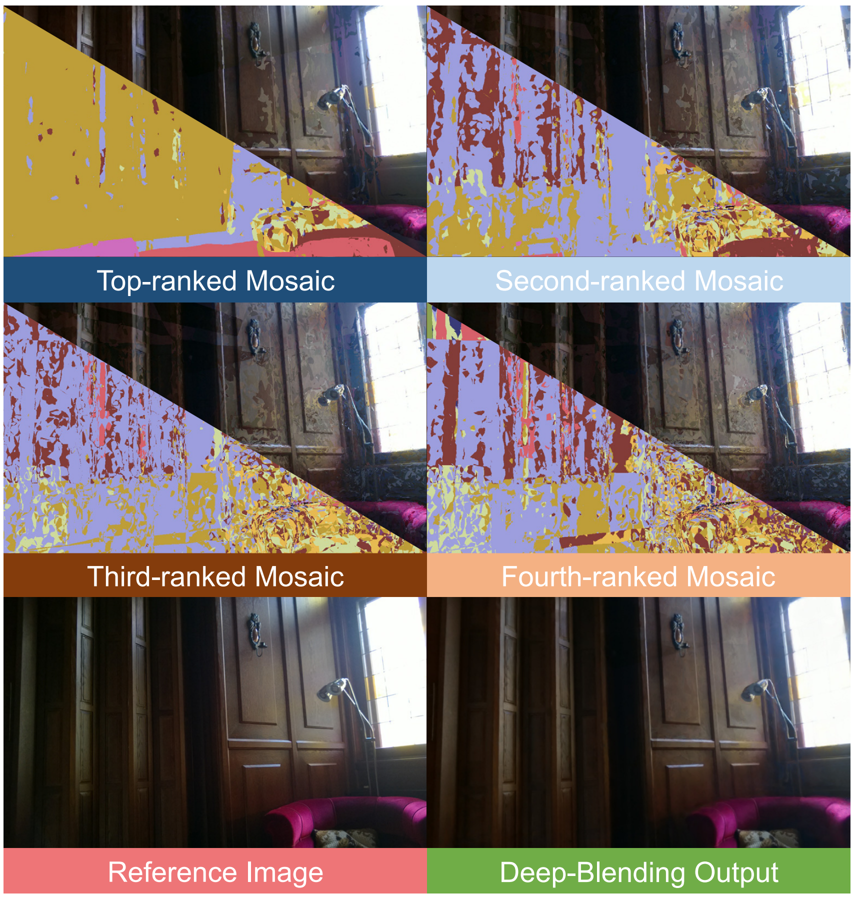


**Figure 6.4:** Our network takes as input ranked mosaics generated from a set of warped candidate views. For each pixel, the candidates are ranked based on their expected blending contribution, and 4 color-image mosaics are formed from the top 4 rankings. Example mosaics are shown in the first two rows. The top right halves show the color mosaic, while the bottom left halves show colormaps of the selection, with each input shown in a different color. Weighted blending outputs from our network (bottom right) are trained by minimizing their difference with real images (bottom left). Our network also blends an RGB view of the global mesh (not shown).

We experimented with using extra input channels (depth, normal, IBR blend weights), but these did not provide a conclusive improvement. This is probably because these inputs have already been used to form the ranked input mosaics, so the extra layers simply contain redundant information already implicitly encoded in the input layers and their relative order.

### 6.2.2 Training data

We train and test our network on a number of different scenes. Note that in our experiments, we show results both with and without images from the test scene being included in the training data. However, the actual results consist only of data samples – the novel viewpoints – that were never seen in training.

In previous work [132, 141], *input* photos are used as ground truth images, in a held-out or leave-one-out strategy for error evaluation. We adopt a similar approach, but to succeed in training our CNN, we require a large number of images with sufficient variety of scene content. For this, we collected a total of 19 scenes from different sources: 7 scenes from Chaurasia et al. [23], 4 scenes from Chapter 5, 1 scene from the Eth3D benchmark [175], and 7 new scenes which we have captured ourselves. Each scene contains between 12 and 418 input images, with resolution varying from $1228 \times 816$ to $2592 \times 1944$. There are 5 indoors scenes, and 5 scenes containing a significant amount of vegetation. In total, we have 2630 images, and we use data augmentation to mitigate the risk of overfitting by taking random 256x256 crops of the images and performing random rotations and horizontal/vertical flips. We use 90% of the images in each scene for training, leaving the remaining 10% as a validation set.

We also experimented with data augmentation for the mosaics themselves, and generated mosaics with blend weights computed for a different camera location. This did not ensure temporal smoothness, but instead suppressed highlights.

### 6.2.3 Training loss

The goal of the training is to determine how well the held-out novel view resembles the input photograph. We face two main challenges: dealing with slight misalignments due to warping and encouraging temporal coherence.

Since our geometry reconstruction is not perfect, we may never obtain an accurate match between the warped input images and the held-out ground truth. In particular, the surface may have slight reconstruction error, and there can be an offset between the warped mosaics and the reference photograph or differences in texture resolution and stretch. Together, these effects mean that we cannot form the reference photograph as a weighted average of the

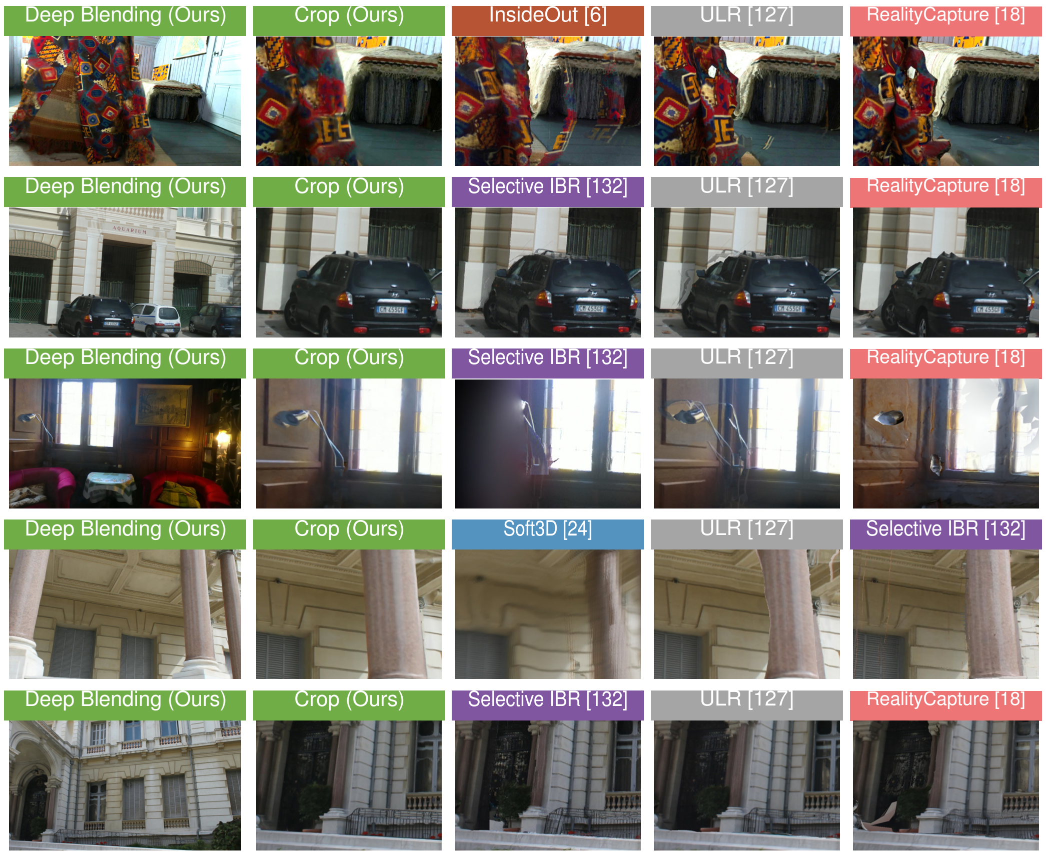


**Figure 6.5:** Results from 5 scenes. **Left:** Full novel view from our solution, followed by a crop using our method. The remaining three columns show previous methods. Five methods are compared: Textured Mesh-based rendering, Selective IBR [132], ULR [127] with soft visibility [129], InsideOut [6], and Soft3D [24]. Only some methods shown per row; videos of all methods are available in the supplemental material. In most cases, errors due to inaccurate geometry result in visual artifacts such as ghosting, incorrect edge reconstruction as well as blur.

input layers. This makes training with an image loss such as $L_1$ or SSIM ill-conditioned. Instead, we build on work developed for style transfer, where “perceptual” image differences have proven their effectiveness, based on pretrained VGG16 network features [169].

For a given novel image $I_N$ and a reference image $I_R$, our loss $\mathscr{L}$ is:

$$\begin{aligned}\mathscr{L}(I_N, I_R) = |I_N - I_R| &+ |\mathrm{VGG16}_{relu12}(I_N) - \mathrm{VGG16}_{relu12}(I_R)| \\ &+ |\mathrm{VGG16}_{relu22}(I_N) - \mathrm{VGG16}_{relu22}(I_R)|\end{aligned} \tag{6.1}$$

where $\mathrm{VGG16}_{relu*}$ are feature activations at different scales of a VGG16 network pretrained on ImageNet. With this formulation, the network is able to create high-quality single frames. However, for difficult regions in the scene, the network struggles to find a good solution and

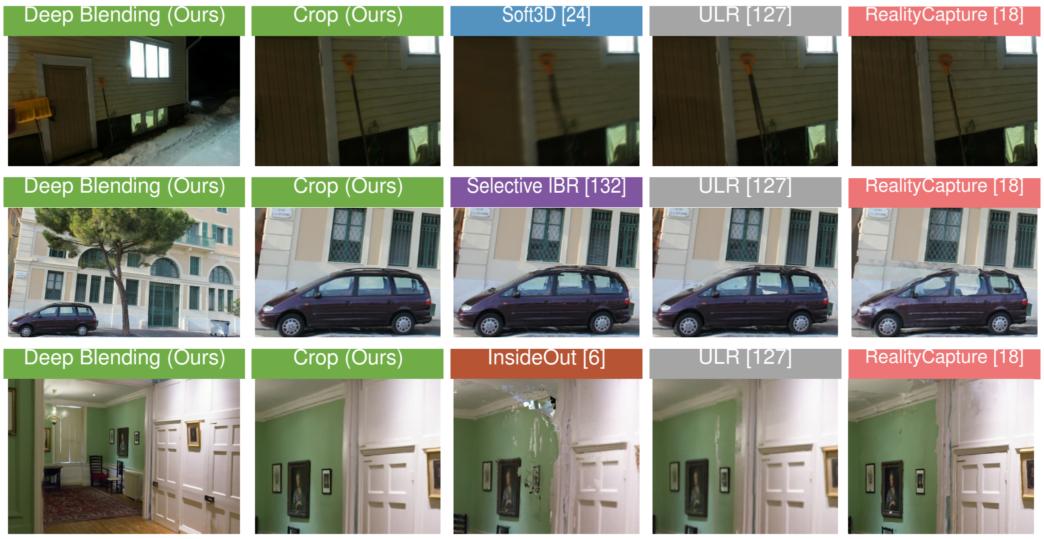


**Figure 6.6:** Results continued, presentation as in Figure 6.5. In some cases, our method can recover from large geometric errors (e.g., car in the middle row).

the weights tend to oscillate from frame to frame. To disambiguate this case, we encourage the network to produce temporally stable results with an auxiliary loss, based on a small window of camera motion. As training data, we generate small 2-frame clips, which start from a reference camera pose and move in a random direction. To keep camera motion relatively small, we set frame-to-frame displacement to be 25% of the average baseline. During training, we run the network twice to generate outputs for both the frames in the clip.

The final temporally consistent loss function we use is as follows:

$$\mathscr{L}_T(I_N, I_R) = \mathscr{L}(I_N^t, I_R) + 0.33 * \mathscr{L}(I_N^{t-1}, \mathscr{W}_f(I_N^t)), \tag{6.2}$$

where $\mathscr{W}_f(I)$ is the warped image $I$ using optical flow. Optical flow is estimated as the average flow of the 4 input mosaics, ignoring global geometry. We can compute this flow exactly for each mosaic, since we have the corresponding geometry and render using OpenGL.

We also tested a loss including image gradients, which in theory should discourage seams due to color harmonization problems. However, this instead encouraged outliers and slowed down convergence. We did not experiment with adversarial losses, which is an interesting future work direction; however, maintaining temporal smoothness would probably be very challenging. We also experimented with a temporal window as input to the network. Interestingly, this did not improve convergence, possibly because the network is unable to correct the parallax between the inputs.

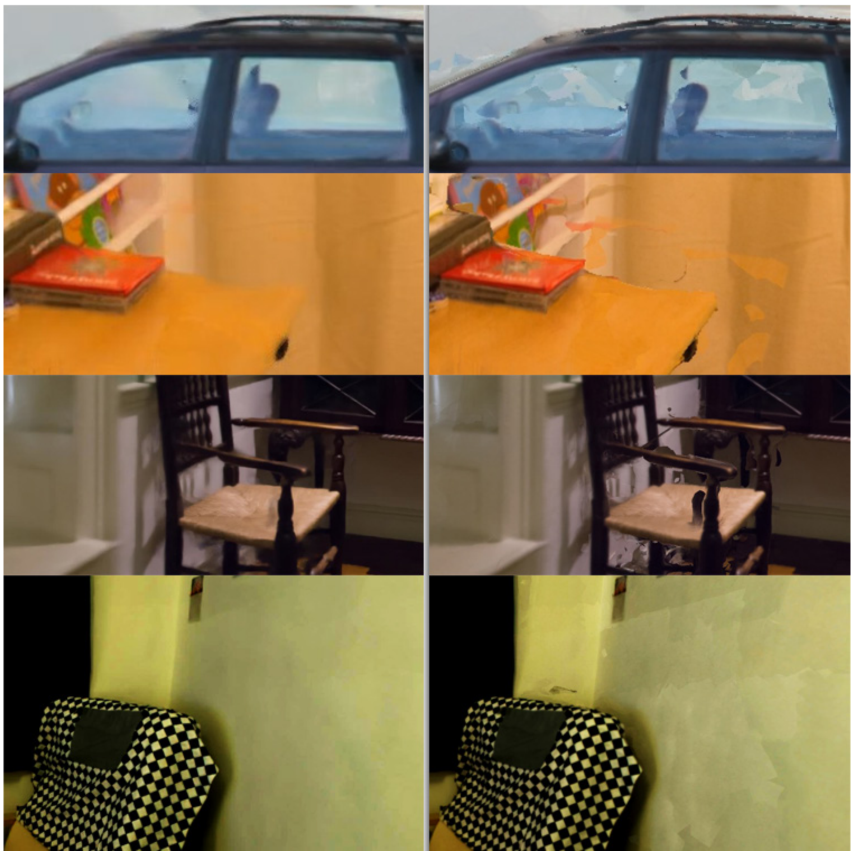


**Figure 6.7:** Our deep blending network vs. the heuristic blend weights used in Chapter 5. **Left:** Our full method, showing the difference in quality due to Deep Blending. **Right:** Heuristic blend weights (see Section 5.3.1).

## 6.3 Implementation and Evaluation

We implemented our system in C++ and OpenGL, combined with the C++ interface of TensorFlow [176] for runtime blend evaluation. Our rendering pipeline is described in Chapter 5.

To maximize the pipeline's potential for interactive rendering times, we implemented custom TensorFlow operations that directly copy the input and output OpenGL textures to and from the network in on-device GPU transfers, making use of the OpenGL/CUDA interop interface available in the CUDA version 9.0 library. A custom CUDA kernel was implemented to increase the speed of 2D spatial upsampling; all other network components used TensorFlow's native operations with cuDNN 7.1 [177].

### 6.3.1 Results and Comparisons

In Figs. 6.5 and 6.6, we show results from 8 scenes, comparing our method with other end-to-end IBR systems:

**RealityCapture:** The textured mesh from [18].

**Selective IBR:** The superpixel IBR approach by [132] using the RealityCapture mesh as input geometry.

**ULR:** Unstructured Lumigraph Rendering [127] with soft visibility [129] and the Reality-Capture mesh as the geometry proxy.

**Soft3D:** The novel view synthesis algorithm by [24], using their custom soft 3D reconstructions.

**InsideOut:** The indoor IBR system by [6] using their custom depth-sensor based 3D reconstructions. Note that this is different from the system described in Chapter 5, which uses improved reconstruction and blending techniques, and does not rely on depth sensor data.

Except for InsideOut, which uses the original reconstructions with depth-sensor data, all methods use the same camera registration and calibration produced by COLMAP. The full set of videos with paths from all these scenes and comparisons with other methods is in supplemental material. We show Soft3D only in 4 scenes, as it has difficulty handling the very unstructured capture of our data and comparisons in more scenes would not be very informative.

Quantitative comparison for IBR is always a difficult endeavor. We provide a quantitative comparison using the Virtual Rephotography approach [141], using the shiftable L1 error metric (see Section 3.3.3). For this, and all following quantitative evaluations, we have used the images from paths of a subset of 5 scenes, namely CreepAttic, Museum-1, NightSnow (rows 1,4 and 6 in Figure 6.5), DrJohnson (row 3 in Figure 6.6), and Hugo-1 (Figure 6.2). It is important to note that this evaluation favors previous methods, since most of the time images captured are quite close to other input views (by definition, to provide overlap for SfM and MVS); our method excels in viewpoints further from the inputs, but meaningful rephotography would require capture of photos specifically for this purpose. The results of this comparison are shown in Figure 6.11.

For the results in Figure 6.5 and 6.6, we included all scenes in the training. We also show results with networks trained without the scene being shown (Figure 6.9). We see that there is little visible difference between the two, which is also confirmed by the numerical results shown in Figure 6.8 (right).

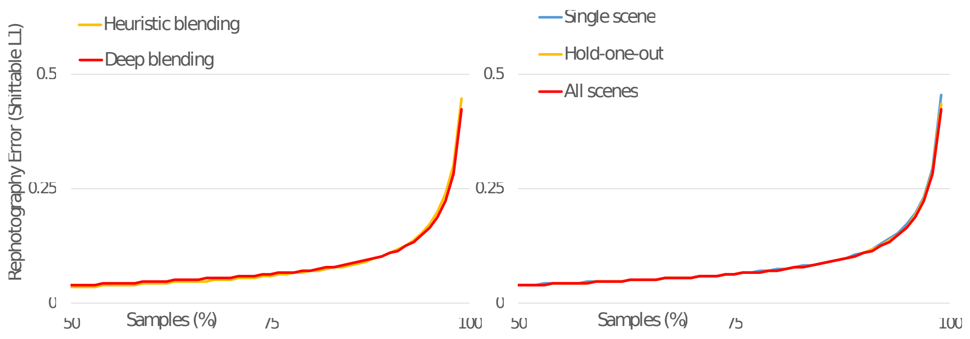


**Figure 6.8:** Rephotography comparison, i. e., how well different approaches can reconstruct held-out input images. The graphs show the cumulative distribution of errors over five scenes (Creepy Attic, Dr Johnson, Museum-1, Hugo-1, and Night Snow), i. e., the percentage of pixels (x-axis) with an error smaller than a threshold (y-axis), starting at the median error. **Left, ablation:** We compare our deep blending with the heuristic blend weights used in Chapter 5. **Right, generalization (evaluated over the same 5 scenes):** We compare a single network trained on all 19 scenes with (1) **hold-one-out** networks that never saw the test scene during training, and (2) **single-scene** networks that only saw the test scene during training.

### 6.3.2 Evaluation of Deep Blending vs. Heuristic Blending

In Figure 6.7, we compare the heuristic blend weights described in Chapter 5 vs. deep blending using the same geometry, isolating the effect of blending only, and observe there are several visible artifacts that are corrected by our solution. Interestingly, this effect is not visibile when comparing the two methods quantitatively with the leave-one-out rephotography [141] results in Figure 6.8. These graphs plot the “shiftable L1 error” (see Section 3.3.3) – the minimum L1 distance for a $7 \times 7$ color patch around each ground-truth image pixel, compared with all output patches that have been shifted up to $\pm 2$ pixels in x and/or y around the source pixel. Interestingly, the performance is similar in the three cases where either or both of our improvements are used. As can be seen in the supplemental material and video, the relative visual importance of each can be scene- and context-dependent, and our complete solution is generally better visually, especially far from the input views.

### 6.3.3 Network Evaluation

We also perform ablation tests on the learning process. In our supplemental material, we provide a large number of videos showing the performance for different ablations in data or method. We have investigated several aspects:

- **Training scenes.** In Figs. 6.8-6.9 and supplemental material, we show the effect of running our network in 5 scenes it did not see during training (hold-one-out). The

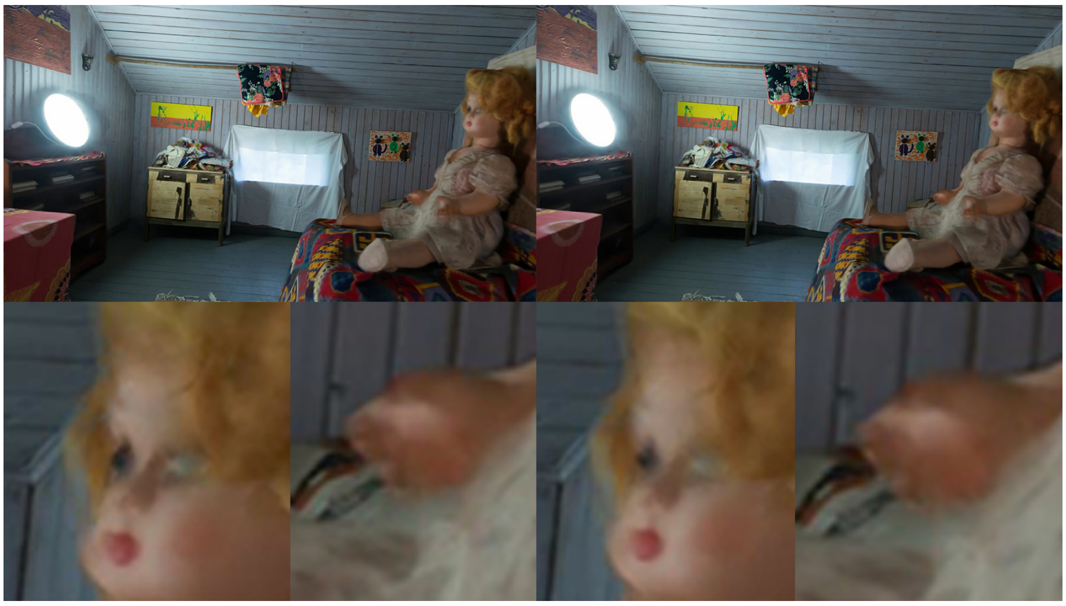


**Figure 6.9: Left:** novel view computed with all scenes used for training, **Right:** same view rendered with a network that never saw the scene during training. The result is similar.

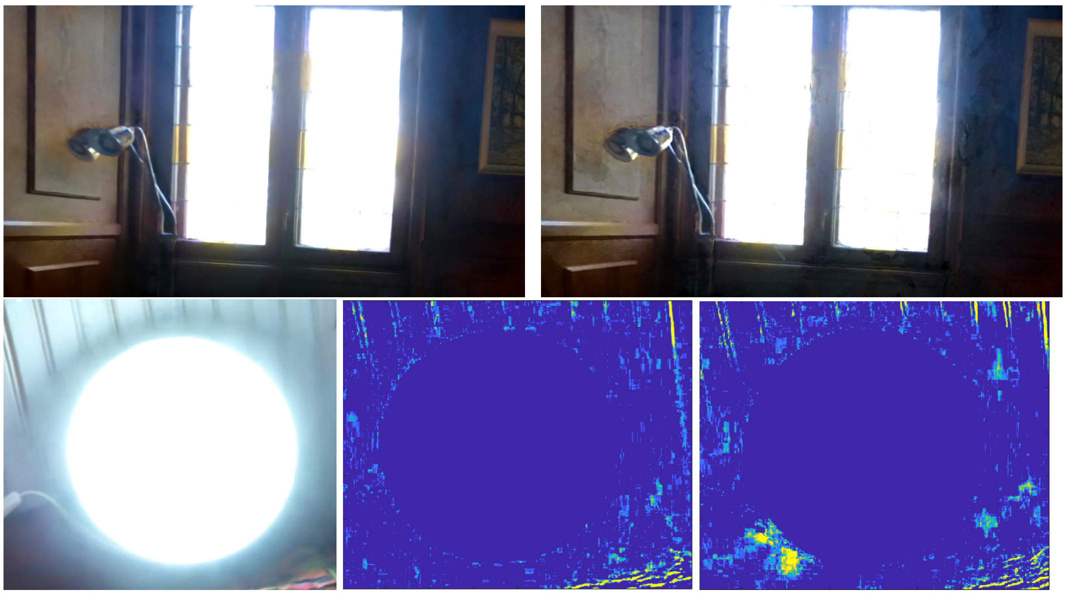


**Figure 6.10: Top Left:** Full method, with the VGG loss. **Top Right:** Same view without the VGG loss. **Bottom Left:** Frame t. **Bottom Middle:** Our full method (10x temporal difference between frame t and t+1) using our full method. **Bottom Right:** Our method, no temporal loss (10x temporal difference between frame t and t+1).

results are similar to training with the scene. We also test the effect of only training with images from one scene (single-scene), which in Figure 6.8 does not show a strong numerical difference, but tends to reduce temporal stability (see supplemental material).

- **Loss.** In Figure 6.10, we show the effect of our perceptual (VGG) loss vs. an $L_1$ loss. Our supplemental videos further show this difference in performance. We also include results of our network trained without the temporal loss; these exhibit strong inter-frame intensity flicker in many of the sequences.

- **Image regression.** We tested a variant of the network that directly regresses (predicts) the output image rather than blend weights. Similarly to [178], we observe that directly regressing the output image works well, but network convergence is much slower.

- **Number of mosaics.** We show supplementary results using only the top 1 or 2 mosaics as network input. We observe diminishing return with more mosaics – each additional mosaic contributes to smaller details, particularly along surface boundaries. Having more mosiacs does, however, lead to smoother view transitions.

- **Training data size.** We show supplemental videos after training with a random 30% subset of our data. Surprisingly, the qualitative performance is fairly constant.

### 6.3.4 Performance

We evaluate the runtime performance of our interactive rendering pipeline at a resolution of $1280 \times 720$ px on a system with a 3.47 GHz Intel Xeon processor and an NVIDIA GTX 1080Ti GPU. (Note that the videos in our supplemental are rendered offline at a resolution of $1920 \times 1080$ px.) Table 6.1 shows average per-frame execution times of our interactive pipeline over several scenes. Runtimes are specific to the scene and are not driven only by the deep blending network component. For smaller scenes, especially indoor scenes, our implementation achieves greater than 30fps performance, which falls off gradually as scene complexity increases. For large outdoor scenes, we observe that the primary bottleneck is in the voxel-wise camera selection, which in our current implementation is tied to a fixed-size spatial subdivision. An adaptive or hierarchical approach is an interesting future direction for this issue.

Preprocessing times for our pipeline varied from scene to scene but generally finished overnight. We treat network training separately, which takes two days, but does not necessarily have to be re-run for each new scene (see Figs. 6.8-6.9). Example processing times for the Creepy Attic scene are shown in Table 6.2.

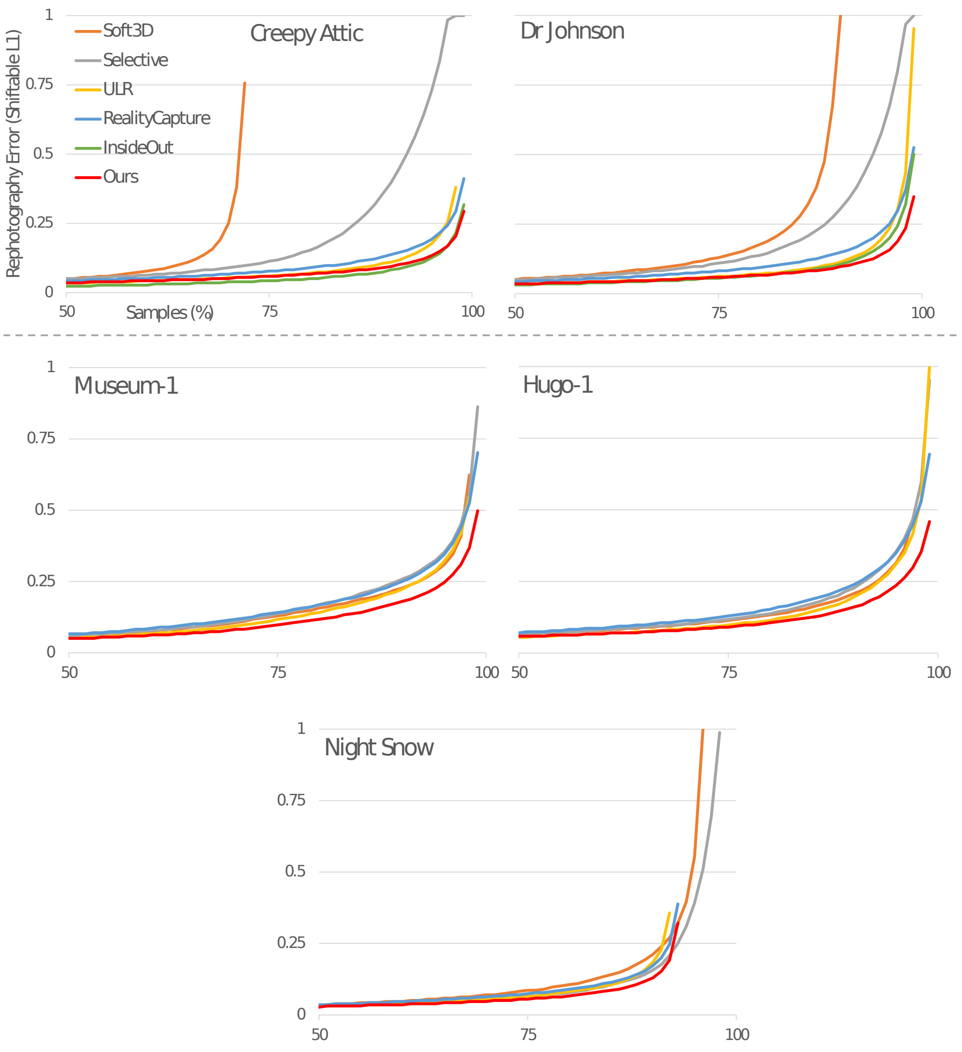


**Figure 6.11:** Rephotography comparison, i. e., how well different approaches can reconstruct held-out input images. With the shiftable L1 difference, lower is better. The graphs show the cumulative distribution of errors, i. e., the percentage of pixels (x-axis) with an error smaller than a threshold (y-axis), starting at the median error. Here, we see that some scenes are harder than others, and that our approach is effective for a higher % of pixels. In this experiment, we only use hold-one-out networks that did not see the test scene during training.

### 6.3.5 Limitations and future work

Our method gracefully degrades when there is very little 3D geometry from the initial 3D reconstruction; we show an example in Figure 6.12. Our method will also tend to flicker when the differences in exposure are very large and inconsistent (e. g., the Bridge scene in the supplemental materials). The network occasionally creates blur, typically in regions with missing geometry or where difficult decisions need to be made (highlights, resolution mismatches etc.). Nonetheless, in the vast majority of the 19 scenes, our method outperforms all previous solutions, although in some cases one artifact is traded for another (e.g., blur instead of seams).

In future work, we would like to address these problems. To reduce blur, one possible avenue would be the use of an adversarial loss [179]; it is unclear how well such an approach would deal with temporal coherence. To reduce flickering for very large exposure inconsistencies would probably require an effective pre-processing step for color harmonization, while respecting differences due to view-dependent materials.

There are several possible avenues of research to improve rendering speed, e. g., using fewer mosaics, more efficient networks architectures [180], model compression [181, 182], or porting the blending network to GLSL [183].

## 6.4 Conclusions

We have extended the system described in Chapter 5 with a deep blending network, demonstrating that it is possible to learn blending weights for viewpoint-free photography while maintaining interactive display rates.

This approach shows promise as a viable solution for room-scale VR experiences. It achieves all the goals from Section 1.1, it works with easy-to-capture photos as input (goal 1), supports free-viewpoint rendering with a large range of motion (goal 2), suppresses artifacts and maintains high-quality results (goal 3). Achieving real-time performance (goal 4), especially in the context of stereo viewing for VR (i. e., at least 90 fps) is possible, but requires performance improvements. We believe this can be addressed with careful code optimization and upgraded hardware: A dual-GPU system would render the views for both eyes in parallel, and next generation GPUs provide further speedups.

**Table 6.1:** Average runtimes (ms/frame) over 100 frames for our IBR system.

| Scene | Non-network Runtime | Total Runtime | Scene | Non-network Runtime | Total Runtime |
|---|---|---|---|---|---|
| Museum-1 | 6.2 | 26.2 | Hugo-1 | 14.1 | 35.6 |
| Creepy Attic | 7.1 | 26.8 | Night Snow | 15.3 | 33.0 |
| Dr Johnson | 12.4 | 33.6 | Boats | 19.7 | 47.6 |

**Table 6.2:** Preprocessing times for Creepy Attic (249 images at $1228 \times 816$) using a 4-core Intel Core i7 processor and an NVIDIA Titan X GPU.

| Component | Runtime | Component | Runtime |
|---|---|---|---|
| COLMAP SfM | 1h | Depth map Refinement | 0.75h |
| COLMAP MVS | 5.25h | Meshing | 1.5h |
| RealityCapture MVS | 0.5h | Network Training | 37h |

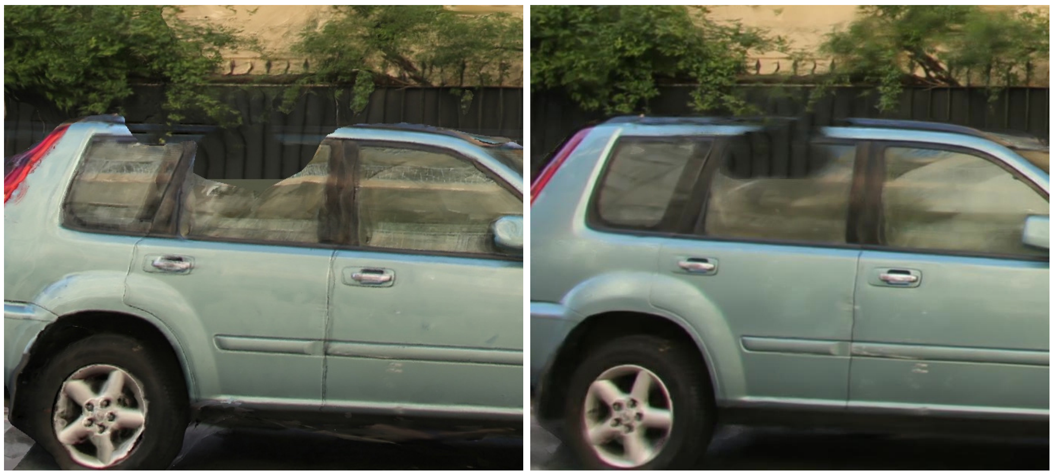


**Figure 6.12: Left:** From the rendering of the mesh, we see that part of the geometry of the car is completely missing. **Right:** Despite significant visual improvement, our method cannot hallucinate the missing geometry.

Chapter 7

# Conclusions and future work

In this thesis, we explored methods to easily capture and digitally preserve a real place, so it can later be revisited realistically in virtual reality (VR). While many recent VR photography methods [160, 9, 10] successfully capture an all around view at a single viewpoint in the scene, our focus is on capturing and reproducing a range of viewpoints in the scene. We call this type of capture *viewpoint-free photography* and investigated the following hypothesis:

> *We can create viewpoint-free photos, suitable for viewing in virtual reality, from casually captured footage, i. e., footage that can be captured in less than 30 minutes using an off-the-shelf camera.*

Specifically, we explored two common VR experiences enabled by modern commodity headsets: (1) seated VR, where the viewpoint is mostly stationary, but the user can peek behind corners, and (2) room-scale VR, where the viewpoint is unconstrained, as the user is expected to freely explore the scene.

We provided evidence that our hypothesis holds true for both VR experiences:

- In Chapter 3, we showed that it's possible to build viewpoint-free photos for seated VR experiences, using easy-to-capture panorama-style footage.
- In Chapter 4, we further investigated capture for seated VR experiences, showing that fast processing times are possible, which holds the promise of delivering on-set feedback — facilitating even easier capture.
- In Chapter 5, we showed that more elaborate capture from multiple viewpoints in the scene enables viewpoint-free photography for room-scale VR experiences.
- In Chapter 6, we deepened our investigation into room-scale VR experiences, showing that quality can be increased by learning how to prevent rendering mistakes from data.

## 7.1 Reaching our goals

While all our methods above provide evidence for our hypothesis, we can gain further insights by revisiting our goals from Chapter 1:

1. ease of capture,
2. sufficient range of motion,
3. high quality (realism), and
4. real-time display rates.

This helps clarify how the methods relate to each other and can point us towards fruitful avenues for future work. We can easily distinguish the methods in the first half of this thesis (Chapters 3-4) from the methods in the latter half (Chapters 5-6) by looking at the range of motion (goal 2) they provide.

In the first half, we target seated VR experiences which require only a small range of motion, placing our focus on easy capture (goal 1). Both approaches in this half achieve this goal by using easy-to-capture panorama footage as input, with the method in Chapter 4 sacrificing some range of motion for even more convenient capture.

The approaches described in the latter half expand the range of motion to support room scale VR experiences. Unsurprisingly, we found that capture for room-scale VR is more elaborate (goal 1), but the resulting viewpoint-free photos contain a higher degree of realism (goal 3), as they preserve some view dependent effects. Compared to Chapter 5, Chapter 6 further improves realism (goal 3) by reducing the number of visible mistakes, albeit with a slight impact on display rates (goal 4).

At the end of the day, the methods presented in this thesis strike a compromise between our four goals. While it is certainly possible that no viewpoint-free photography system can simultaneously maximize all goals, we believe our methods are not yet *Pareto-optimal*: improvements can still be made towards each goal without sacrificing the others. We discuss a few such improvements below.

### 7.1.1 Ease of capture

Modern 360 cameras, such as the Ricoh Theta[1] or the Insta360[2], could drastically reduce the number of input images, as they capture a much larger view of the scene compared to

[1]https://theta360.com
[2]https://www.insta360.com/

normal photos. Implicitly, the methods presented in this thesis assume crisp, noise-free images without rolling shutter distortion. This makes capture tedious in low-light conditions, e. g., indoor scenes and overcast days, requiring patience and a steady hand. A system that remains robust to this type of noise, blur and distortion would facilitate truly casual capture, enabling the user to quickly and carelessly capture viewpoint-free photos by simply waving a camera around in the scene.

### 7.1.2 Range of motion

Currently, we've used panorama-style input footage only for the creation of seated VR experiences. Foreseeably, it would be possible to extend the range of motion for these experiences by hallucinating the unobserved regions in the scene. This could be done using inpainting techniques, which are able to fill in both geometry [184] and colors [163, 185] using the captured footage as context. Advances in this line of work would lead towards bridging the gap between seated and room-scale VR experiences.

We can also imagine extending our room-scale captures to arbitrary scales, such as city-wide or even world-wide VR experiences. This begs the question on how to maintain real-time display rates, considering that the data won't fit on any consumer GPU, and maybe not even in working memory, suggesting that streaming approaches may be necessary [186, 187]. Capturing sufficient data for this type experience is also problematic, certainly not casual, and more than a single person can do alone. Viewpoint-free photography systems at this scale would have to look into crowd-sourcing capture, and develop reconstruction approaches that are robust to this type of large-scale heterogeneous data.

### 7.1.3 High quality

Our viewpoint-free photos preserve view-dependent effects by blending between highlights in the captured footage. This works well for rough highlights that have been captured sufficiently, but is often unconvincing for sharp highlights on mirror-like surfaces. This could be addressed with an approach that explicitly solves and recreates the motion of highlights on surfaces in the scene [22]. We have not addressed viewpoint-free photography for transparent objects: a formidable problem which would require rethinking both the reconstruction [188] and rendering components [24] of our methods.

The approaches developed in this thesis are robust to slight scene motion and can ignore occasional dynamic objects that move throughout the scene. However, our methods are designed to capture *places* rather than *moments* and struggle with large changes in scene appearance or a large amount of moving objects, such as a crowd of people. While multi-camera rigs trivially enable viewpoint-free photos of moments, an approach that works with casually captured footage is likely more elaborate, since it would need to consolidate input images that represent different points in time.

### 7.1.4 Real-time display rates

While all approaches in this thesis facilitate real-time display rates, we never pay close attention to the hardware which is used for rendering. This comes to a head in Chapter 6, where a modern dual-GPU computer would be needed to reach this goal. However, many current and future headsets work with much less powerful hardware — especially untethered headsets[3][4][5] designed for the rooms-scale experiences we explore in Chapters 5-6. Maintaining real-time display rates on such limited hardware poses an interesting challenge. This opens up many exciting avenues for future work, e. g., more efficient acceleration structures, network compression [181] or precomputation [19, 20].

## 7.2 Beyond our goals

The work in this thesis concerns capture and display of viewpoint-free photos. As discussed above, there are several further improvements to be made towards the goals we stated in Chapter 1. However, we feel that viewpoint-free photography also opens up research opportunities outside these goals.

**Video.** A large portion of the footage captured today has a moving object in the foreground - especially the footage we share with others. What if we could capture viewpoint-free video of an event with family and friends? While viewpoint-free video for room-scale VR is still an unexplored problem, commercial efforts use custom-built capture rigs to address this problem for seated VR experiences [31].

**Editing.** Today, we have access to a large collection of tools for editing our photographs. This ranges from simple color correction to more elaborate operations, such as object removal [163] and seamlessly compositing many images together [189]. How can we bring these operations to the realm of viewpoint-free photography?

**Augmented Reality.** Viewpoint-free photos contain a significant amount of the visual information in a scene. Can we use this for augmented reality applications, i. e., augment and

[3]`https://www.microsoft.com/en-us/hololens`
[4]`https://www.magicleap.com/`
[5]`https://www.oculus.com/quest/`

edit real images in the scene? For example, this could help resolve visibility when inserting virtual objects into the scene [190]

## 7.3 Concluding remarks

In this thesis, we provided evidence that viewpoint-free photography is feasible from casually captured data. Using footage from off-the-shelf cameras or smartphones, our methods create high-quality viewpoint-free photos suitable for both seated and room-scale VR experiences. There is a lot of interesting future work left to explore, including further improvements towards the goals we stated in Chapter 1, but also research outside the scope of this thesis, such as editing, augmented reality and capturing moving scenes.

Viewpoint-free photography facilitates many exciting applications, e. g., stay-at-home tourism, virtual tours for real-estate, and immersive preservation for places of cultural importance or personal significance. In a foreseeable future, hopefully within the next five years, we may even see some of these applications become available to consumers, as our field begins to move beyond prototypes and towards a commercial viewpoint-free photography system, just as photography and videography before it.

Appendix A

# Supplemental material

This appendix contains links to the supplemental material associated with each chapter.

## A.1 Casual 3D Photography

This is the supplemental material associated with Chapter 3.

- Supplemental video:

  `https://youtu.be/wGBistgOsyQ`

- Supplemental material:

  `http://vis.cs.ucl.ac.uk/Download/G.Brostow/Casual3D/index.html`

- Datasets:

  `http://visual.cs.ucl.ac.uk/pubs/casual3d/datasets.html`

## A.2 Instant 3D Photography

This is the supplemental material associated with Chapter 4.

- Supplemental video:

  `https://youtu.be/OlgVoplUJ8g`

- Supplemental material:

  `http://visual.cs.ucl.ac.uk/pubs/instant3d/supplemental`

## A.3 Scalable Free-Viewpoint Image-Based Rendering

This is the supplemental material associated with Chapter 5. Listed as “Heuristic Blending”.

- Supplemental material:

  `http://team.inria.fr/graphdeco/deep-blending`

- Datasets:

  `http://visual.cs.ucl.ac.uk/pubs/insideout/datasets.html`

## A.4 Deep Blending for Free-Viewpoint Image-Based Rendering

This is the supplemental material associated with Chapter 6.

- Supplemental video:

  `https://youtu.be/F1g7PAZ9cI4`

- Supplemental material:

  `http://team.inria.fr/graphdeco/deep-blending`

- Datasets:

  `http://repo-sam.inria.fr/fungraph/deep-blending/data`

## Appendix B

# Casual 3D Photography: Plane-sweep MVS implementation

**Plane sweep schedule:** We distribute $N = 220$ depth planes between $d_{min} = 0.5$ m and $d_{max} = 1500$ m according to the square root of their disparities ($\propto \sqrt{z^{-1}}$), striking a balance between uniform depth intervals ($\propto z$) which undersamples foreground depths and uniform disparity intervals ($\propto z^{-1}$) which in turn oversamples nearby regions with sub-centimeter accuracy.

**Smoothness Term $E_{smooth}(i, j)$:** Our smoothness energy

$$E_{smooth}(i,j) = w_{color}(c_i, c_j)\; w_{depth}(d_i, d_j) \tag{B.1}$$

encourages the resulting depth map to be smooth wherever the image lacks texture. It is a weighted sum over all neighboring pixels $(i, j)$, where

$$w_{color}(c_i, c_j) = \alpha_c + (1 - \alpha_c) e^{\frac{-|c_i - c_j|^2}{2\sigma_c^2}} \tag{B.2}$$

down-weighs the smoothness at strong image gradients, and

$$w_{depth} = \frac{\left|d_i - d_j\right| + \alpha_d \min\left(\left|d_i - d_j\right|,\ \tau_d\right)}{Z} \tag{B.3}$$

is a piece-wise linear smoothness energy on depth labels, which grows faster near zero. In all experiments, we set $\alpha_c = 0.05$, $\sigma_c = 0.033$, $\tau_d = 16$ and $\alpha_c = 31$. We normalize $w_{depth}$ to be between 0 and 1 using $Z = N(\alpha d \tau_d + 1) = 715$.

**Photo-consistency Term $E_{photo}(i)$:** To evaluate photoconsistency at each depth plane, we reproject a pixel $i$ into 16 neighboring images, where we use sum-of-absolute-differences (SAD) to compare both colors ($SAD_c$) and image gradients ($SAD_\nabla$). We use the transformation $f(x) = 1 - \exp(-x/\sigma_{photo})$ to truncate both SAD costs and measure the photoconsistency against one of the neighbor images $n$ as the weighted average $\alpha_\nabla f(SAD^n_\nabla) + (1 - \alpha_\nabla) f(SAD^n_c)$. We then account for occlusions in the scene by computing the overall photoconsistency by averaging the 50% lowest photoconsistency energies from all neighbor images. In all experiments, we set $\alpha_\nabla = 0.9$ and $\sigma_{photo} = 0.033$.

Finally, we filter each slice of the resulting cost volume using

1. A $5 \times 5$ Shiftable matching window [191], and
2. A bilateral filter [192, 193] with a 10px standard deviation and $\sigma_c = 0.033$.

In general, the bilateral filter smooths the cost while preserving important objects edges [193]. However, in our experiments we found that this filter is sensitive to large color variations and can make the filter footprint very small for small patches of unusual color. This is why we apply shiftable-matching windows, essentially prefiltering the cost volume across such small patches.

**Photo-consistency Confidence $w_{photo}(i)$:** Photo-consistency measures are less reliable when a pixel is visible in a small number of views: With more views, repeating patterns are no longer ambiguous. Textureless surfaces also become less ambiguous, as the textured parts in each view rule out locations where such surfaces cannot be.

To avoid mistakes for pixels visible in few views, we limit the influence of the photo-consistency term for such pixels, and rely more on the smoothness term to resolve their depths. Concretely, for each pixel $i$, we try all $N$ depth hypotheses, and record $M_i$, the maximum number of views that could see it. We then compute the confidence

$$w_{photo}(i) = w_{min} + (1 - w_{min})\,\text{clamp}_{[0,1]}\left(\frac{M_i - M_{min}}{M_{max} - M_{min}}\right), \tag{B.4}$$

a linear ramp between 1 for reliable pixels seen in $M_{max} = 8$ or more images, to a minimum confidence $w_{min} = 0.1$ for pixels seen in $M_{min} = 4$ images or fewer.

Appendix C

# Instant 3D Photography: Implementation Details

## C.1 Color harmonization

As described in Section 4.2.3.3, we convert the images to the channel-decorrelated CIELAB color space, and then process each channel independently. We solve a linear system to compute global affine color-channel adjustments (a scale $\alpha$ and an offset $\beta$) for each source image, such that the adjusted color values in the overlapping regions agree as much as possible.

Formally, for each channel we minimize the sum

$$E = \lambda_{pairwise} E_{pairwise} + \lambda_{reg} E_{reg} \tag{C.1}$$

of a pairwise color alignment term $E_{pairwise}$ (normalized by $\lambda_{pairwise}$) and a regularization term $E_{reg}$ (strength $\lambda_{reg} = 0.33$).

**Pairwise term** We compute our pairwise term as sum over every pair of overlapping images $A, B$ in the panorama,

$$E_{pairwise} = \sum_{A,B} \sum_{p_i, p_j \in C(A,B)} ||\alpha_A * p_A^i + \beta_A - \alpha_B * p_b^i - \beta_B||^2, \tag{C.2}$$

where $\alpha_A$ and $\alpha_B$ are the scale terms for images A and B, $\beta_A$ and $\beta_B$ are the offset terms, and $C(A, B)$ is a regularly subsampled set of corresponding pixels between the images A and B. To make the system better conditioned, we do not include over-saturated pixels (luminance $> \tau_{saturated} = 0.98$)

To make sure that the relative strength between the pairwise term and the regularization term does not differ too much, we normalize the pairwise term by the total number of correspondences. In other words,

$$\lambda_{pairwise} = \frac{1}{\sum_{A,B} |C(A,B)|}. \tag{C.3}$$

**Regularization term** Our regularization term,

$$E_r = \sum_A ||\alpha_a - 1||^2 + ||\beta_a||^2 \tag{C.4}$$

encourages identity transformations and prevents the trivial solution of scaling everything down to 0.

**Over-saturated pixels** Since our geometry fusion prevents saturated pixels at all costs, any remaining over-saturated regions in the panorama are likely over-exposed in all of the input images. In this case, applying color harmonization dulls bright highlights. We preserve highlights by computing the final image as a soft blend between the original and the harmonized image, where the threshold above (luminance $> \tau_{saturated}$) ensures that we only use the original image for over-saturated highlights.

## C.2 Feathering

We reduce visible seams in the final panorama with feathering, i.e. instead of making a hard choice for the color $c_p$ of a pixel $p$, we compute a weighted sum

$$c_p = \frac{\sum w_p^\alpha c_p^\alpha}{\sum w_p^\alpha} \tag{C.5}$$

of the colors associated with all available labels $\alpha$.

Seams most often occur at 1) label boundaries 2) input image boundaries. We account for both of these issues by computing the label weights $w_p^\alpha$ as the minimum of:

1. A soft label mask, i.e. a $50 \times 50$ box-blurred version of the label mask, and
2. a soft image mask, i.e. a function which linearly decreases from 1 to 0 in regions that are close to image boundaries (within 15% of the input image diagonal).

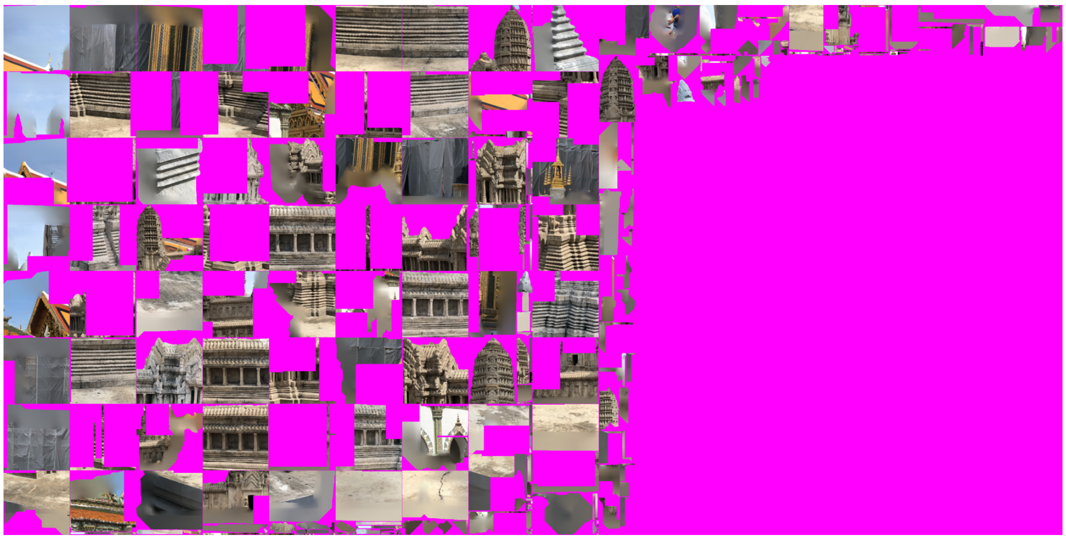


**Figure C.1:** Texture atlas for the ANGKOR WAT MINIATURE scene.

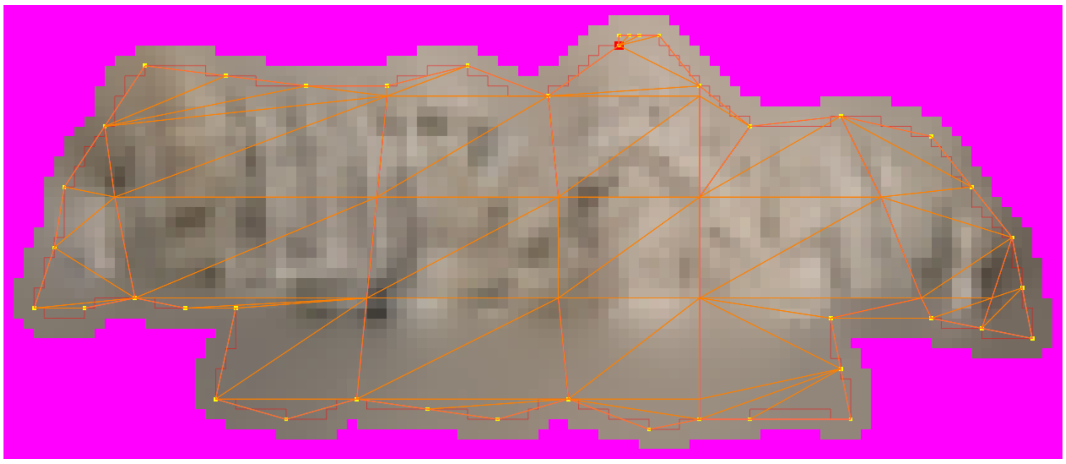


**Figure C.2:** Mesh simplification is performed in 2D on a per-chart basis.

## C.3 Mesh Processing

The meshes generated by the algorithm in Section 4.2.4 use a prohibitively large triangle count, since we have vertices for every pixel in the stitched panorama.

The first step to reducing the size is to decompose the mesh into charts that are not self-overlapping in the sense that they contain multiple vertices that occupy the same pixel position, but at different depth. We build the charts using a flood-fill like algorithm that propagates from a seed vertex as long as certain conditions are satisfied (not self-overlapping, chart size thresholds, etc.) All the charts are packed into a texture atlas (Figure C.1).

Next, we generate simplified meshes directly for each chart as described below. The algorithm below operates in 2D in the texture atlas chart domain. The final mesh is obtained by projecting the vertices into world space according to their image position and depth. We simplify the outline of the chart using the Ramer-Douglas-Peucker algorithm [194], making sure that chart boundaries that are shared between two charts are simplified in the same way. Next, we add vertical "stud" edges that are spaced and subdivided according to the desired tessellation level (the near-vertical edges in Figure C.2). Finally, we generate a mesh for the chart by first decomposing it into monotone polygons, and then triangulating each monotone polygon [195].

# Bibliography

[1] Johannes Peter Kopf, Lars Peter Johannes Hedman, and Richard Szeliski. Three-dimensional scene reconstruction from set of two dimensional images for consumption in virtual reality, July 31 2018. US Patent App. 10/038,894.

[2] Devin Coldewey. How facebook's new 3D photos work. https://techcrunch.com/2018/06/07/how-facebooks-new-3d-photos-work. Accessed: 2019-03-25.

[3] CR-Play. http://www.cr-play.eu. Accessed: 2016-10-15.

[4] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3D photography. *ACM Trans. Graph.*, 36(6):234:1–234:15, 2017.

[5] Peter Hedman and Johannes Kopf. Instant 3D photography. *ACM Trans. Graph.*, 37(4):101:1–101:11, 2018.

[6] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM Trans. Graph.*, 35(6):231:1–231:11, 2016.

[7] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 37(6):257:1–257:15, 2018.

[8] Forte VFX1 Headgear Virtual Reality System User's Manual. http://archive.org/details/forte-technologies-vfx1-headgear-virtual-reality-system. Accessed: 2018-10-4.

[9] Shmuel Peleg, Moshe Ben-Ezra, and Yael Pritch. Omnistereo: panoramic stereo imaging. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(3):279–290, 2001.

[10] Christian Richardt, Yael Pritch, Henning Zimmer, and Alexander Sorkine-Hornung. Megastereo: Constructing high-resolution stereo panoramas. *CVPR*, pages 1256–1263, 2013.

[11] Sony Cyber-shot User Guide - 3D Sweep Panorama. `https://helpguide.sony.net/gbmig/44240861/v1/eng/contents/02/03/09/09.html`. Accessed: 2018-10-4.

[12] Google. Carboard Camera. `https://googleblog.blogspot.com/2015/12/step-inside-your-photos-with-cardboard.html/`, 2015. Accessed: 2016-12-26.

[13] Immersive 3D Spaces for real-world applications — Matterport. `http://matterport.com/`. Accessed: 2016-10-15.

[14] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH*, 1993.

[15] Daniel G Aliaga, Thomas Funkhouser, Dimah Yanovsky, and Ingrid Carlbom. Sea of images. In *Vis.*, pages 331–338. IEEE, 2002.

[16] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(8):1362–1376, 2010.

[17] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. *ECCV*, 2016.

[18] CapturingReality RealityCapture. RealityCapture. `https://capturingreality.com`, 2016. Accessed: 2018-10-01.

[19] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH*, pages 31–42. ACM, 1996.

[20] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *SIGGRAPH*, pages 43–54. ACM, 1996.

[21] S. N. Sinha, D. Steedly, and R. Szeliski. Piecewise planar stereo for image-based rendering. In *ICCV*, pages 1881–1888. IEEE, 2009.

[22] Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. Image-based rendering in the gradient domain. *ACM Trans. Graph.*, 32(6):199:1–199:9, 2013.

[23] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graph.*, 32(3):30:1–30:12, 2013.

[24] Eric Penner and Li Zhang. Soft 3D reconstruction for view synthesis. *ACM Trans. Graph.*, 36(6):235, 2017.

[25] Yasutaka Furukawa and Carlos Hernández. Multi-view stereo: A tutorial. *Found. Trends. Comput. Graph. Vis.*, 9(1-2):1–148, 2015.

[26] Heung-Yeung Shum, Shing-Chow Chan, and Sing Bing Kang. *Image-based rendering*. Springer, 2008.

[27] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press.

[28] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[29] Ryan S. Overbeck, Daniel Erickson, Daniel Evangelakos, Matt Pharr, and Paul Debevec. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *ACM Trans. Graph.*, 37(6):197:1–197:15, 2018.

[30] Robert Anderson, David Gallup, Jonathan T. Barron, Janne Kontkanen, Noah Snavely, Carlos Hernandez Esteban, Sameer Agarwal, and Steven M. Seitz. Jump: Virtual reality video. *ACM Trans. Graph.*, 35(6), 2016.

[31] Facebook. RED and Facebook Collaborate to Create the World's First Cinematographic 6DOF Camera. `https://facebook360.fb.com/2018/05/01/red-facebook-6dof-camera/`, 2016. Accessed: 2018-10-14.

[32] Joel Carranza, Christian Theobalt, Marcus A. Magnor, and Hans-Peter Seidel. Free-viewpoint video of human actors. *ACM Trans. Graph.*, 22(3):569–577, 2003.

[33] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. In *ACM Trans. Graph.*, volume 23, 2004.

[34] C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor. Virtual video camera: Image-based viewpoint navigation through space and time. *Comp. Graph. Forum*, 29(8):2555–2568, 2010.

[35] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34(4):69:1–69:13, 2015.

[36] C Lawrence Zitnick and Sing Bing Kang. Stereo for image-based rendering using image over-segmentation. *Int. J. Comput. Vision*, 75(1):49–65, 2007.

[37] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, July 2006.

[38] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *ICCV*, pages 72–79. IEEE, Sept 2009.

[39] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M Seitz. Multi-view stereo for community photo collections. In *ICCV*, pages 1–8. IEEE, 2007.

[40] Kaan Yücer, Changil Kim, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. Depth from gradients in dense light fields for object reconstruction. In *3DV*, October 2016.

[41] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *arXiv:1607.02565*, 2016.

[42] T. Bertel, N. D. F. Campbell, and C. Richardt. Megaparallax: Casual 360° panoramas with motion parallax. *IEEE TVCG*, 2019. to appear.

[43] Christopher G. Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, pages 147–151, 1988.

[44] Jianbo Shi and Carlo Tomasi. Good features to track. *CVPR*, pages 593 – 600, 1994.

[45] David G Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Visio*, 60(2):91–110, 2004.

[46] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *ECCV*, pages 404–417. Springer, 2006.

[47] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[48] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):756–777, 2004.

[49] Alessandro Chiuso, Roger Brockett, and Stefano Soatto. Optimal structure from motion: Local ambiguities and global estimates. *Int. J. Comput. Vision*, 39(3):195–228, 2000.

[50] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[51] Changchang Wu. Towards linear-time incremental structure from motion. In *3DV*, pages 127–134. IEEE, 2013.

[52] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016.

[53] Pierre Moulon, Pascal Monasse, and Renaud Marlet. Adaptive structure from motion with a contrario model estimation. In *ACCV*, pages 257–270. Springer, 2013.

[54] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *arXiv preprint arXiv:1610.06475*, 2016.

[55] P Debevec, Y Yu, and G Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Workshop*, pages 105–116. Springer, 1998.

[56] Christoph Rhemann Michael Bleyer and Carsten Rother. Patchmatch stereo - stereo matching with slanted support windows. In *BMVC*, pages 14.1–14.11, 2011.

[57] Vladimir Tankovich, Michael Schoenberg, Sean Ryan Fanello, Adarsh Kowdle, Christoph Rhemann, Maksym Dzitsiuk, Mirko Schmidt, Julien Valentin, and Shahram Izadi. Sos: Stereo matching in o (1) with slanted support windows. In *IROS*, pages 6782–6789, 2018.

[58] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *J. Mach. Learn. Res.*, 17(1):2287–2318, 2016.

[59] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *CVPR*, pages 66–75, 2017.

[60] David Gallup, Jan-Michael Frahm, Philippos Mordohai, Qingxiong Yang, and Marc Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *CVPR*, pages 1–8. IEEE, 2007.

[61] Silvano Galliani, Katrin Lasinger, and Konrad Schindler. Massively parallel multiview stereopsis by surface normal diffusion. In *ICCV*, pages 873–881. IEEE, 2015.

[62] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440. IEEE, 2015.

[63] Aasma Hosni, Christoph Rhemann, Michael Bleyer, Carsten Rother, and Margrit Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(2):504–511, 2013.

[64] Richard Szeliski. Image alignment and stitching: A tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, 2006.

[65] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23(3):673–678, 2004.

[66] Kaiming He, Jia Sun, and Xiaoou Tang. Guided image filtering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(6):1397–1409, 2012.

[67] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. Constant time weighted median filtering for stereo matching and beyond. In *ICCV*, pages 49–56, 2013.

[68] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):65–81, 2004.

[69] Heiko Hirschmuller. Stereo vision in structured environments by consistent semi-global matching. In *CVPR*, pages 2386–2393. IEEE, 2006.

[70] Dilip Krishnan and Richard Szeliski. Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph.*, 30(6):177:1–177:10, 2011.

[71] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. DeepStereo: Learning to predict new views from the world's imagery. In *CVPR*, pages 5515–5524. IEEE, June 2016.

[72] Andrej Karpathy, Justin Johnson, and Others. Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io`, 2019. Accessed: 2019-03-17.

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*. IEEE, June 2016.

[74] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*. IEEE, 2018.

[75] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *Int. J. Rob. Res.*, 32(11):1231–1237, 2013.

[76] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CVPR*, 2017.

[77] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *GCPR*, pages 31–42. Springer, 2014.

[78] K. Pulli, H. Abi-Rached, T. Duchamp, L. G. Shapiro, and W. Stuetzle. Acquisition and visualization of colored 3d objects. In *ICPRPR*, pages 11–15. IEEE, 1998.

[79] O. Hall-Holt and S. Rusinkiewicz. Stripe boundary codes for real-time structured-light range scanning of moving objects. In *ICCV*, pages 359–366. IEEE, 2001.

[80] Damien Lefloch, Rahul Nair, Frank Lenzen, Henrik Schäfer, Lee Streeter, Michael J. Cree, Reinhard Koch, and Andreas Kolb. *Technical Foundation and Calibration Methods for Time-of-Flight Cameras*, pages 3–24. Springer, Berlin, Heidelberg, 2013.

[81] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CVPR*, 2017.

[82] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *NIPS*, pages 2366–2374, 2014.

[83] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *CVPR*. IEEE, 2018.

[84] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. In *ECCV*, pages 740–756. Springer, 2016.

[85] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312. ACM, 1996.

[86] Michael Goesele, Brian Curless, and Steven M Seitz. Multi-view stereo revisited. In *CVPR*, pages 2402–2409. IEEE, 2006.

[87] Katja Wolff, Changil Kim, Henning Zimmer, Christopher Schroers, Mario Botsch, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. Point cloud noise and outlier removal for image-based 3d reconstruction. In *3DV*, pages 118–127, 2016.

[88] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, pages 127–136. IEEE, 2011.

[89] Simon Fuhrmann, Fabian Langguth, and Michael Goesele. Mve: A multi-view reconstruction environment. *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage (GCH '14)*, pages 11–18, 2014.

[90] OpenMVS. Openmvs: open multi-view stereo reconstruction library. `https://github.com/cdcseacave/openMVS`, 2016. Accessed: 2016-12-26.

[91] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *SIGGRAPH*, pages 311–318. ACM, 1994.

[92] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH*, 21(4):163–169, 1987.

[93] Patrick Labatut, Jean-Philippe Pons, and Renaud Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *ICCV*, pages 1–8. IEEE, 2007.

[94] Michal Jancosek and Tomas Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. *CVPR*, pages 3121–3128, 2011.

[95] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In *SGP*. The Eurographics Association, 2006.

[96] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, 2013.

[97] Qi Shan, Brian Curless, Yasutaka Furukawa, Carlos Hernandez, and Steven M. Seitz. Occluding contours for multi-view stereo. In *CVPR*, pages 4002–4009. IEEE, 2014.

[98] George Vogiatzis, Carlos Hernández Esteban, Philip H. S. Torr, and Roberto Cipolla. Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(12):2241–2246, 2007.

[99] Vu Hoang Hiep, Renaud Keriven, Patrick Labatut, and Jean-Philippe Pons. Towards high-resolution large-scale multi-view stereo. In *CVPR*, pages 1430–1437. IEEE, 2009.

[100] COLMAP. Colmap. `https://colmap.github.io`, 2017. Accessed: 2018-01-20.

[101] Daniel N Wood, Daniel I Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H Salesin, and Werner Stuetzle. Surface light fields for 3D photography. In *SIGGRAPH*, pages 287–296. ACM, 2000.

[102] Krzysztof Wolski, Daniele Giunchi, Nanyang Ye, Piotr Didyk, Karol Myszkowski, Radoslaw Mantiuk, Hans-Peter Seidel, Anthony Steed, and Rafał K. Mantiuk. Dataset

and metrics for predicting local visible differences. *ACM Trans. Graph.*, 37(5):172:1–172:14, 2018.

[103] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*. IEEE, 2018.

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

[105] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graph.*, 23(3):294–302, 2004.

[106] Fan Zhang and Feng Liu. Parallax-tolerant image stitching. *CVPR*, pages 3262–3269, 2014.

[107] F. Perazzi, A. Sorkine-Hornung, H. Zimmer, P. Kaufmann, O. Wang, S. Watson, and M. Gross. Panoramic video from unstructured camera arrays. *Comp. Graph. Forum*, 34(2):57–68, 2015.

[108] Kaimo Lin, Nianjuan Jiang, Loong-Fah Cheong, Minh N. Do, and Jiangbo Lu. SEAGULL: seam-guided local alignment for parallax-tolerant image stitching. *ECCV*, pages 370–385, 2016.

[109] Hiroshi Ishiguro, Masashi Yamamoto, and Saburo Tsuji. Omni-directional stereo for making global map. In *ICCV*, pages 540–547. IEEE, 1990.

[110] Shmuel Peleg and Moshe Ben-Ezra. Stereo panorama with a single camera. *CVPR*, pages 395–401, 1999.

[111] Fan Zhang and Feng Liu. Casual stereoscopic panorama stitching. *CVPR*, pages 2002–2010, 2015.

[112] Facebook. Facebook Surround 360. `https://facebook360.fb.com/facebook-surround-360/`, 2016. Accessed: 2016-12-26.

[113] Robert Konrad, Donald G. Dansereau, Aniq Masood, and Gordon Wetzstein. SpinVR: towards live-streaming 3D virtual reality video. *ACM Trans. Graph.*, 36(6):article no. 209, 2017.

[114] Sheng-Jie Luo, I-Chao Shen, Bing-Yu Chen, Wen-Huang Cheng, and Yung-Yu Chuang. Perspective-aware warping for seamless stereoscopic image cloning. *ACM Trans. Graph.*, 31(6):article no. 182, 2012.

[115] Sunghoon Im, Hyowon Ha, François Rameau, Hae-Gon Jeon, Gyeongmin Choe, and In So Kweon. All-around depth from small motion with a spherical panoramic camera. *ECCV*, pages 156–172, 2016.

[116] Ke Colin Zheng, Sing Bing Kang, Michael F. Cohen, and Richard Szeliski. Layered depth panoramas. *CVPR*, pages 1–8, 2007.

[117] Jayant Thatte, Jean-Baptiste Boin, Haricharan Lakshman, and Bernd Girod. Depth augmented stereo panorama for cinematic virtual reality with head-motion parallax. *ICME*, 2016.

[118] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. In *SIGGRAPH*, pages 299–306. ACM, 1999.

[119] Benno Heigl, Reinhard Koch, Marc Pollefeys, Joachim Denzler, and Luc Van Gool. Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *Mustererkennung 1999*, pages 94–101. Springer, 1999.

[120] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. *Comp. Graph. Forum*, 31(2):305–314, 2012.

[121] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM Trans. Graph.*, 35(6), 2016.

[122] Pratul P Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. Learning to synthesize a 4D RGBD light field from a single image. In *ICCV*, volume 2, page 6. IEEE, 2017.

[123] Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! Large-scale texturing of 3D reconstructions. In *ECCV*, pages 836–850. Springer, 2014.

[124] Qian-Yi Zhou and Vladlen Koltun. Color map optimization for 3D reconstruction with consumer depth cameras. *ACM Trans. Graph.*, 33(4):155:1–155:10, 2014.

[125] Bastian Goldlücke, Mathieu Aubry, Kalin Kolev, and Daniel Cremers. A super-resolution framework for high-accuracy multiview reconstruction. *Int. J. Comput. Vision*, 106(2):172–191, 2014.

[126] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *SIGGRAPH*, pages 371–378. ACM, 2001.

[127] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *SIGGRAPH*, pages 425–432. ACM, 2001.

[128] Kari Pulli, Hugues Hoppe, Michael Cohen, Linda Shapiro, Tom Duchamp, and Werner Stuetzle. *View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data*, pages 23–34. Springer, 1997.

[129] Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson De Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. Floating textures. *Comp. Graph. Forum*, 27(2):409–418, 2008.

[130] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH*, pages 279–288. ACM, 1993.

[131] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH*, pages 39–46. ACM, 1995.

[132] Rodrigo Ortiz-Cayon, Abdelaziz Djelouah, and George Drettakis. A Bayesian approach for selective image-based rendering using superpixels. In *3DV*, pages 469–477. IEEE, 2015.

[133] Michael Goesele, Jens Ackermann, Simon Fuhrmann, Carsten Haubold, Ronny Klowsky, Drew Steedly, and Richard Szeliski. Ambient point clouds for view interpolation. *ACM Trans. Graph.*, 29(4):95:1–95:6, 2010.

[134] Andrew Fitzgibbon, Yonatan Wexler, and Andrew Zisserman. Image-based rendering using image-based priors. *Int. J. Comput. Vision*, 63(2):141–151, 2005.

[135] Oliver Woodford and Andrew W Fitzgibbon. Fast image-based rendering using hierarchical image-based priors. In *BMVC*, volume 1, pages 260–269, 2005.

[136] Oliver J Woodford, Ian D Reid, Philip HS Torr, and Andrew W Fitzgibbon. Fields of experts for image-based rendering. In *BMVC*, volume 3, pages 1109–1108, 2006.

[137] Oliver J Woodford, Ian D Reid, and Andrew W Fitzgibbon. Efficient new-view synthesis using pairwise dictionary priors. In *CVPR*, pages 1–8. IEEE, 2007.

[138] Steven M. Seitz and Charles R. Dyer. View morphing. In *SIGGRAPH*, pages 21–30. ACM, 1996.

[139] Christian Lipski, Felix Klose, and Marcus Magnor. Correspondence and depth-image based rendering: a hybrid approach for free-viewpoint video. *IEEE (T-CSVT)*, 24(6):942–951, Jun 2014.

[140] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *ECCV*, pages 286–301. Springer, 2016.

[141] Michael Waechter, Mate Beljan, Simon Fuhrmann, Nils Moehrle, Johannes Kopf, and Michael Goesele. Virtual rephotography: Novel view prediction error for 3d reconstruction. *ACM Trans. Graph.*, 36(1):article no. 8, 2017.

[142] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH*, 1998.

[143] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004.

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

[145] Robert T. Collins. A space-sweep approach to true multi-image matching. In *CVPR*, pages 358–363. IEEE, 1996.

[146] Nikos Komodakis and Georgios Tziritas. Approximate labeling via graph cuts based on linear programming. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(8):1436–1453, 2007.

[147] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Trans. Graph.*, 26(3), 2007.

[148] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Trans. Graph.*, 23(3):689–694, 2004.

[149] Frederic Besse, Carsten Rother, Andrew Fitzgibbon, and Jan Kautz. Pmbp: Patch-match belief propagation for correspondence field estimation. *Int. J. Comput. Vision*, 110(1):2–13, 2014.

[150] Simon Fuhrmann and Michael Goesele. Floating scale surface reconstruction. *ACM Trans. Graph.*, 33(4):article no. 46, 2014.

[151] Benjamin Ummenhofer and Thomas Brox. Global, dense multiscale reconstruction for a billion points. *ICCV*, 2015.

[152] Hyowon Ha, Sunghoon Im, Jaesik Park, Hae-Gon Jeon, and In So Kweon. High-quality depth from uncalibrated small motion clip. *CVPR*, 2016.

[153] Jonathan T. Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(8):1670–1687, 2015.

[154] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *ECCV*, pages 1–14, 2010.

[155] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(5):815–830, 2010.

[156] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISSAPP)*, pages 331–340, 2009.

[157] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`, 2017. Accessed: 2018-10-01.

[158] Nicholas Ayache. *Vision Stéréoscopique et Perception Multisensorielle: Application à la robotique mobile*. Inter-Editions (MASSON), 1989.

[159] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Comput. Graph. Appl.*, 21(5):34–41, 2001.

[160] Julio Zaragoza, Tat-Jun Chin, Michael S. Brown, and David Suter. As-projective-as-possible image stitching with moving dlt. *CVPR*, pages 2339–2346, 2013.

[161] M. Jancosek and T. Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR*, pages 3121–3128. IEEE, 2011.

[162] Xiaoyan Hu and Philippos Mordohai. A quantitative evaluation of confidence measures for stereo vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2121–2133, 2012.

[163] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24, 2009.

[164] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. Constant time weighted median filtering for stereo matching and beyond. In *ICCV*, pages 49–56. IEEE, 2013.

[165] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH*, pages 209–216. ACM, 1997.

[166] Ola Olsson and Ulf Assarsson. Tiled shading. *J. Graphics, GPU, and Game Tools*, 15(4):235–251, 2011.

[167] Wenfeng Li and Baoxin Li. Joint conditional random field of multiple views with online learning for image-based rendering. In *CVPR*. IEEE, 2008.

[168] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241. Springer, 2015.

[169] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, pages 694–711. IEEE, 2016.

[170] Ke Colin Zheng, Alex Colburn, Aseem Agarwala, Maneesh Agrawala, David Salesin, Brian Curless, and Michael F Cohen. Parallax photography: Creating 3D cinematic effects from stills. In *GI*, pages 111–118. Canadian Information Processing Society, 2009.

[171] Johannes Kopf, Michael F Cohen, and Richard Szeliski. First-person hyper-lapse videos. *ACM Trans. Graph.*, 33(4), 2014.

[172] Sergi Pujades, Frédéric Devernay, and Bastian Goldluecke. Bayesian view synthesis and image-based rendering principles. In *CVPR*, pages 3906–3913. IEEE, 2014.

[173] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[174] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. 1, 2017.

[175] Thomas Schöps, Johannes L. Schönberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *CVPR*. IEEE, 2017.

[176] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[177] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. In *Proceedings of the NIPS Workshop on Deep Learning and Representation Learning*, 2014.

[178] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graph.*, 36(4):97, 2017.

[179] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*. IEEE, July 2017.

[180] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint abs/1704.04861*, 2017.

[181] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint abs/1611.06440*, 2016.

[182] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint abs/1511.06530*, 2015.

[183] Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. Deep shading: Convolutional neural networks for screen-space shading. 36(4), 2017.

[184] M. Firman, O. M. Aodha, S. Julier, and G. J. Brostow. Structured prediction of unobserved voxels from a single depth image. In *CVPR*, pages 5431–5440. IEEE, 2016.

[185] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. IEEE, 2016.

[186] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *I3D*, pages 15–22. ACM, 2009.

[187] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *MobiSys*, pages 151–165. ACM, 2015.

[188] Bojian Wu, Yang Zhou, Yiming Qian, Minglun Gong, and Hui Huang. Full 3d reconstruction of transparent objects. *ACM Trans. Graph.*, 37(4):103:1–103:11, 2018.

[189] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.

[190] Aleksander Holynski and Johannes Kopf. Fast depth densification for occlusion-aware augmented reality. *ACM Trans. Graph.*, 37(6):194:1–194:11, 2018.

[191] Aaron F. Bobick and Stephen S. Intille. Large occlusion stereo. *Int. J. Comput. Vision*, 33(3):181–200, 1999.

[192] Kuk-Jin Yoon and In-So Kweon. Locally adaptive support-weight approach for visual correspondence search. In *CVPR*, volume 2, pages 924–931. IEEE, 2005.

[193] Christoph Rhemann, Asmaa Hosni, Michael Bleyer, Carsten Rother, and Margit Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, pages 3017–3024. IEEE, 2011.

[194] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244—-256, 1972.

[195] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.