Augmenting visual memories

Aleksander Holynski

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2022

Reading Committee:

Steven M. Seitz, Chair

Richard Szeliski, Chair

Brian Curless, Chair

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Augmenting visual memories

Aleksander Holynski

Chair of the Supervisory Committee:

We often rely on photographs and other forms of physical and digital media for capturing, preserving, and sharing the visual experiences that we encounter throughout our lives. While many common forms of media (like photographs) are able to effectively capture and reproduce detailed representations of our visual experiences, much of what we remember from these experiences is usually not captured in a photograph.

When compared to our (incredibly rich) visual memories, photographs often lack a lot of crucial contextual information that is necessary to our understanding a particular event, e.g., details that help us answer questions like "where did this take place?", "what led to this moment?", and "what happened next?". Unlike human memories, which include information about our unbounded, three-dimensional, and ever-changing surroundings, photographs only capture a fixed field-of-view and a fixed point in time. Although recent technological developments have brought about a number of ways for capturing more detailed, immersive, and realistic depictions of our visual memories (such as 3D cameras and 360-degree videos), the vast majority of our past memories were captured by older technologies, such as the still photograph. As such, these memories remain permanently preserved in outdated forms of media, limiting in the realism, immersion, and fidelity with which they are able to portray our past experiences.

In this thesis, I investigate methods for elevating this type of legacy imagery to more modern and immersive visual experiences by recovering or synthesizing the context that was lost during the capture process. First, I provide a survey of modern forms of media, specifically

focusing on those which provide more immersion or visual fidelity than the traditional still photograph. I also review methods for augmentation of legacy captures, i.e., methods that turn older, less immersive content into modern forms of media. Finally, I present three novel techniques for enabling automatic augmentation of captured memories: (1) a method for automatically turning captured photos into high-quality video sequences; (2) a method for 3D reconstruction of previously captured smartphone camera videos, enabling different forms of 3D interaction, like virtual object insertion and light manipulation; and (3) a method for large-scale 3D structure from motion targeted towards handheld smartphone footage, which enables 3D reconstruction and visualization of large scenes, like buildings and city blocks that would otherwise be difficult to capture.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

My journey through the Ph.D. was not particularly straightforward. At many points over the years, I questioned my choices, my abilities, and even my sanity. The works in this thesis, and indeed my completion of the Ph.D., is owed entirely to the support, encouragement, and inspiration I've received from an enormous collection of people. I am incredibly grateful to all of them, and would like to acknowledge them in approximate chronological order.

Early years In high school, during countless disciplinary visits to her office, our principal Jacqueline Kelley encouraged me to redirect my rebellious teenage energy towards academics. My undergraduate research mentors at the University of Illinois, Robin Kravets and Svetlana Lazebnik, gave me my first opportunity to engage in independent research. During a shared internship at Qualcomm in San Diego, Tomasz Trzciński was the first to encourage me to get involved in computer vision. Ardalan Khosrowpour and Igor Fedorov, my first research collaborators, showed me a first glimpse of the research process.

The Ph.D. I'm incredibly grateful to have had Steve Seitz, Rick Szeliski, and Brian Curless as my Ph.D. advisors and role models. I entered the Ph.D. largely unprepared to perform independent research, and spent much of my early years learning the necessary skills. I commend my advisors' patience throughout this process, and am indebted to their guidance and support. I deeply cherish the time I have spent working with them.

I am also very grateful for the collection of collaborators I've had over the years: Chris Sweeney, Cecilia Zhang, Xiuming Zhang, Yifan Wang, Johannes Kopf, David Geraghty, Jan Michael Frahm, and Jeong Joon Park. Their insights, guidance, support, and company were key pieces in my continued excitement and motivation for work.

Skills & Passions Over the course of the Ph.D., I discovered new passions, learned new skills, and gained new insights. I attribute each of these to the influence of my outstanding colleagues, collaborators, and mentors. Specifically...

Teaching: At the University of Illinois, David Forsyth's course on optimization methods showed me that a technical course can be taught in a way that is both informative *and* entertaining. His lectures were full of colorful anecdotes describing the scientific process, motivating arguments about algorithmic decisions, and engaging storytelling. At the University of Washington, Brian Curless' computer graphics class served as an example of the importance of interaction and creative expression in the learning process. Brian's class was full of demos of fun visual phenomena as well as projects that gave students the freedom to exercise

their creative interests and personal motivations. My excitement for teaching was further fueled by my experience lecturing for the Computer Vision class taught by Steve Seitz, Rick Szeliski, and Harpreet Sawhney. I'm very thankful for having been given this opportunity.

Presenting: Michal Irani's invited talk at the University of Washington in my early years of the Ph.D. has since served as my gold-standard for a research presentation. I distinctly remember that each slide in her presentation seemed to answer whatever questions had popped into my head the slide before. Ever since, I follow this same approach whenever putting together a presentation, i.e., by simulating and addressing the thoughts that I imagine an audience might have while watching my talk. Steve Seitz's presentation style has also (unsuprisingly) greatly influenced my own. In particular, his focus on visuals, his use of informal and fluid communication, and his use of evocative phrases like *"Picture yourself in..."* and *"Imagine if we could do X"* — these have all become common tropes in my own presentations.

Writing: Throughout my Ph.D., I've used Rick Szeliski's book on Computer Vision, as well as his past papers, as a guide for academic writing. In fact, when writing papers, I often find myself deliberately imitating the tone and style that I've learned to be Rick's. My excitement and passion for writing is also largely attributed to Rick's stories about the time he spent writing his book on Lake Wenatchee. His telling of this experience encouraged me to think of writing as a *fun* task, something I hadn't done before. My father, Marek Hołyński, and my uncle, Nader Tehrani, also both played a role in my aspiration to be a good writer, with my father having published several of his own books on computer graphics and artificial intelligence, and my uncle frequently publishing eloquent articles in architecture journals.

Mentorship: I first began to appreciate the process of mentoring students during my time as a teaching assistant for the UW Virtual Reality Capstone course led by Steve Seitz and Ira Kemelmacher. Steve and Ira were kind enough to include me in advisory meetings for the student projects, and encouraged me to provide feedback and engage in the mentoring process. Both in this case, and in my future mentoring experiences, a great portion the advice I gave and the suggestions I made were borrowed from advice I had previously received from Steve, Rick, and Brian.

Deliberate research: I have learned an enormous amount about the practices that lead to reliable research progress from Rick Szeliski, Kevin Matzen, Clement Godard, and Peter Hedman (although, admittedly, I often forget to implement them!). I've always admired Rick's careful note-taking and documentation of his research ideas — a practice I try to emulate whenever I can. Rick's insistence on visualizing intermediate outputs and starting with simple experiments has also had a very large impact on my research productivity. During my time at Facebook, Kevin Matzen's step-by-step, careful scientific experimentation left an impression on me as being the *"correct"* way of doing research. Clement Godard (and his incredible attention to detail) has been a constant positive influence, forcing me to thoroughly inspect my training data, triple-check my pixels, and make sure that I blur before resizing. Peter Hedman's influence has made me a much stronger debugger, reminding me to ask myself

"What's the simplest set of experiments that conclusively identify the problem?". In my early years at the University of Washington, Richard Newcombe and Rick Szeliski taught me to never ignore the details: to carefully understand the underlying mathematics when working on a high-level research problem, and never to treat low-level concepts as "black boxes".

Problem selection: While I'm far from an expert at picking the right research problems, as can be noted by the numerous scrapped projects during my Ph.D., any of the insights I have about choosing good research projects come from Steve Seitz. When choosing a new problem, I typically start by asking myself the usual collection of Steve Seitzian questions, like *"So...what's the killer demo?"*.

Engineering: During my summer internship at Google, I learned a lot from Changchang Wu and Carlos Hernandez-Esteban about engineering and writing clean and reusable code. This learning continued back at the University of Washington, while I was working with Chris Sweeney (the postdoc with the best coding practices I've ever seen). The code I wrote with Chris has continued to be useful for many years afterwards.

Optimism: While I have always been an (almost excessively) optimistic person, Johannes Kopf helped me realize this same sense of optimism for research. The year I spent working with him at Facebook taught me to think of every problem as solvable — given enough time, excitement, caffeine, and the right combination of ideas.

Reviewing: Over the course of many long walks around Lake Union, Jan Michael Frahm shared his philosophies on the academic process. From him, I learned to see negative reviews not as insults but rather unique opportunities for improvement. I also learned that being a good reviewer is offering the type of constructive feedback and guidance that I would want myself.

Deep learning: I spent the early years of my Ph.D. (foolishly) insisting on avoiding deep learning, despite the insistence of Clement Godard & Keunhong Park. I am very grateful to them, and also Steve Seitz, Xuan Luo, and Peter Hedman for convincing me to not be afraid of changing research areas. I especially appreciate the time that Keunhong and Clement spent answering all my questions during the early stages of my first deep learning project.

Balance: Rick Szeliski and Chris Sweeney served as my models for work-life balance, showing me that it's possible to be successful at work and still maintain time for personal hobbies and interests. I still don't know where they find the time, but I'll figure it out someday.

I owe my sanity throughout the Ph.D. to my friends...

...from before the Ph.D.: Sammy Nammari, Igor Fedorov, Chinmay Deshmukh, Kristie Matsuoka, Yesenia Roman, Haydee Casellas, Sebastian Sagramoso, Paul Garrido, Eduardo Korb.

...in the Ph.D.: Harley Montgomery, Joe Redmon, Doug Woos, Camille Cobb, Jeff Snyder, Daryl Zuniga, Ryan Maas, John Toman.

...made in Seattle: Rucha Wad, Rachel Choi, Wenjun Wu, Daphne Liang, Nan Wang, Jason Estepan, Michelle Curran, Mira Slavcheva, Svet Kolev, Haley Cho, Elise DeGoede Dorough, Jie Li, Eileen Kim, Yashraj Narang, Fanny Huang, Michal Clark, Johannes Linder, Thida Myint, Kei Kase, Amy Chen, Ula Jun, Alireza Rezaei, Rosanna Bellaville, Dominik Moritz, Lilian de Greef, Hessam Bagherinezhad, Daniel Gordon, Nacho Cano, Cleo Hage, Koosha Khalvati, and Sabrina Li.

...made during summer internships: Tal Remez, Clement Godard, Peter Hedman, Cecilia Zhang, Wolf Kienzle, Stan Manilov, Matthias Springer, Fabian Langguth, Joel Janai, Suhib Alsisan, Gal Eichenboim, David Geraghty, Daniel Scharstein.

...in the GRAIL lab at UW: Roy Or-El, Yifan Wang, Keunhong Park, Xuan Luo, Kostas Rematas, Soumyadip Sengupta, Xiaojuan Wang, Luyang Zhu, Jeong Joon Park, Qi Shan, Ricardo Martin Brualla, Aditya Sankar, Edward Zhang, Chung-Yi Weng, Isaac Tian, Adam Fishman, Supasorn Suwajanakorn, Jingwei Ma, Johanna Karras.

In the last two years of my Ph.D., during the COVID-19 pandemic, the majority of my (virtual) interactions were with a small group of people: Keunhong Park, Clement Godard, Xuan Luo, Peter Hedman, Wenjun Wu. Our frequent calls, brainstorming sessions, and virtual social events helped make two years of lockdown feel much less lonely.

Lastly, and most importantly, I attribute my life choices, my continued success, the good bits of my personality, and my passions to my parents Fariba Tehrani and Marek Holynski, and the rest of my family, Nader Tehrani, Ahmad Tehrani, and Parvaneh Tehrani, whose accomplishments, even in the face of overwhelming hardship, inspired me to always try harder and do better.

Chapter 1

INTRODUCTION

The visual world around us is incredibly rich with information, comprised of things such as color, texture, shading, reflections, and transparency. As humans, we perceive all this information by collecting and interpreting the light rays from our surroundings. Along with other sensory signals, this information defines our experiences: our perception of the ever-changing state of the world. When we find them important enough, these captured experiences are subsequently committed to memory, so as to preserve details about a moment of particular sentimental, cultural, or historical significance. Unfortunately, our biological memory banks are imperfect, and as time passes, most of the memories kept in our heads inexorably fade. As such, for centuries and millenia, people have collectively relied on other forms of physical or digital media for capturing, preserving, and sharing these memories.

Typically, these forms of media operate as an intermediate stage in our visual system (Fig. 1.1): they observe the real world in our place and store these observations in a representation that can be transmitted to our eyes at a later time. While this process is effective at capturing and communicating experiences, it unfortunately seldom occurs without a loss

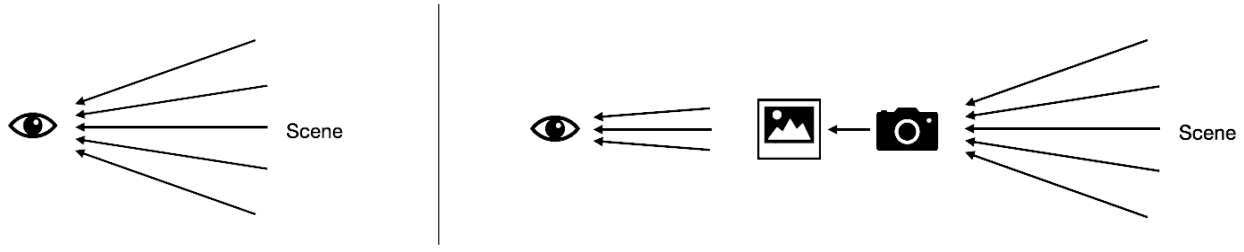


Figure 1.1: The visual system. Our eyes perceive the world by accumulating and interpreting the light rays emitted from our surroundings (left). A camera (right) operates as an intermediate step in this process, by capturing a scene and preserving it in an image, which can be viewed by our eyes at any future point in time.

of information. As an example, the still photograph (the most common form of visual media today) is able to capture an immense amount of information about a scene, storing millions of pixels that can describe even the most minute textural details. Nevertheless, still images fail to truly capture the full experience of observing a particular scene. This is because the capture process discards crucial contextual information such as *spatial context*, information from outside the camera's field-of-view and beyond its line of sight, as well as *temporal context*, information about what led to that moment and how the scene is changing. Look back at your most recent photograph: most of the feelings and details that you remember from this experience are likely not captured in that image.

Since the invention of still photography in the early 1800s, new hardware and software solutions have been developed to capture this missing context, such as videos, which capture temporal context, or 360-degree cameras, which can capture images without restrictions on the field of view. Still, although newer forms of media are able to capture more immersive and detailed depictions of the physical world, there remains an issue: not all memories have been captured with these new technologies. In fact, the vast majority were (and still are) captured with older technologies, such as still-photography cameras, and are therefore permanently preserved in outdated forms of media, lacking much of the necessary context for portraying realistic, immersive, and high-fidelity depictions of these past experiences.

This begs the question: can this discarded context be recovered? In this thesis, I focus on answering this question by exploring methods for *augmenting* our captured memories: methods that convert older forms of media into newer, higher fidelity, and more realistic depictions of the world by re-introducing the information that was lost during the capture process. In effect, these methods mimic the natural process of context recovery that humans undergo when viewing an image. Subtle cues in the image, combined with our prior knowledge of the world and our imagination, can be used to hallucinate context: motion, textural details, setting, or occluded objects. This process is highly subjective, and depends on the viewer's past experiences and associations (Figure 1.2).

Unfortunately, while this process is natural to humans, performing this task computa-

Figure 1.2: Context: localized & transient. Our prior knowledge of the world can be used to extract context from an image, allowing us to identify the setting of the first image as a barber shop. Unfortunately, this prior knowledge is not universal, as it depends on our unique observations of a diverse and evolving world. As such, our (modern) priors are less effective for the second image, since the setting depicted is no longer commonplace. Most would guess it to also be from a barber shop (the most visually similar modern scene), but it actually depicts a dentistry practice from 1929. In another hundred years, the first image may have also lost much of its implied context.

tionally, *i.e.*, automatically recovering and rendering the missing context, is significantly more challenging since computers lack the complex priors that humans have learned from years of observing the world. Learning these complex priors is an ongoing goal of computer vision and artificial intelligence research, and although great strides have been made in recent years with the advent of deep learning, this problem is far from solved. Towards this goal, this thesis demonstrates a number of techniques for augmenting visual memories by automatically recovering and re-introducing two types of discarded information: *temporal* context and *spatial* context.

Augmenting temporal context Still photographs only capture a single, fixed moment in time. This representation is starkly different from our memories of experiences, which seldom capture only a frozen snapshot of the world. Instead, our experiences contain *temporal context*: information about the changing state of the world. Temporal context can help answer questions like:

Figure 1.3: Animating images. Given a single input image (a), our method estimates how the scene was moving when the picture was captured (b), and uses that motion to convert the image into a looping video (c).

Motion:	How are things in the scene moving?
Evolution:	How do object appearances change over time?
Past:	What led to this moment?
Future:	What is likely to happen next?

Existing methods for adding temporal context to images do so by injecting synthesized motion or changes to appearance, thus converting the image into an animated video. These *image animation* techniques can be used to increase the immersion of a still photograph, e.g., by adding moving ripples to water, environmental effects like rain, or clouds blowing in the wind. Unfortunately, the vast majority of these techniques are limited in realism, due to the difficulty of the problem: predicting the motion and appearance of objects in the future is a vastly under-constrained task. In Chapter 4, we demonstrate a solution to this problem: a method for *automatic* and *realistic* image animation that can turn images of natural scenes into seamlessly looping videos with realistic motion and texture (Fig. 1.3).

While videos clearly contain more temporal information than images, they too have their limitations. Just as images are snapshots of *single* moments in time, videos are snapshots of several, i.e., they only capture a fixed set of instances. Videos are both finite (they contain an endpoint) and sparse, with a limited sampling frequency. As such, there are types of

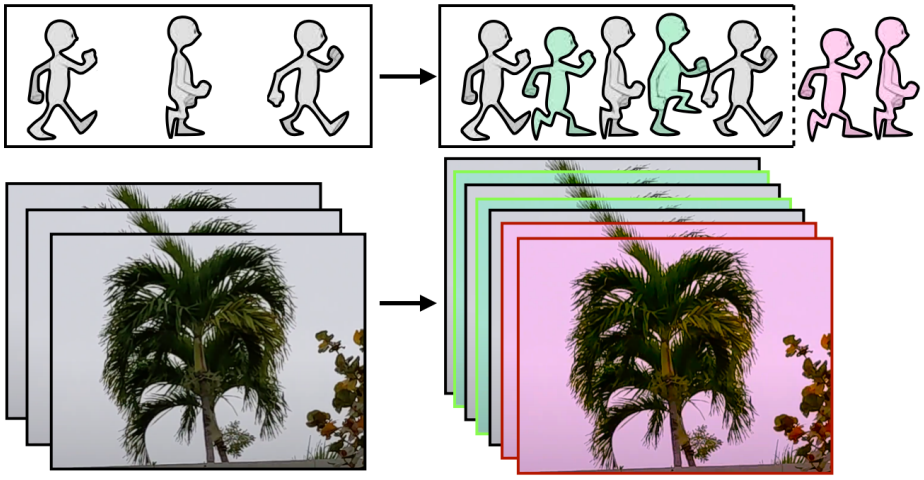


Figure 1.4: Temporal augmentation of videos. Videos consist of a series of sparse observations of the world (frames), distributed in time. Given an input video with periodic motion, such as a tree waving in the wind, our method can increase the number of samples, either by interpolating motion (green) or extrapolating motion (red), producing a higher framerate or longer video as a result.

temporal information that simply cannot be captured in some videos, such as very low or high frequency motions. In Chapter 4.2, we discuss a method for recovering *additional* context for videos, by either increasing the temporal sampling rate or extending the videos in time (Fig. 1.4).

Augmenting with spatial context Still photographs and videos are limited to predetermined viewpoints with fixed fields-of-view, both of which are decided upon at the time of capture and unalterable afterwards. Compared to the way we experience the world, this representation lacks *spatial context*: the information we gain by having binocular vision (a pair of eyes) and the ability to move our heads, walk around, and explore the world. Spatial context helps answer the questions:

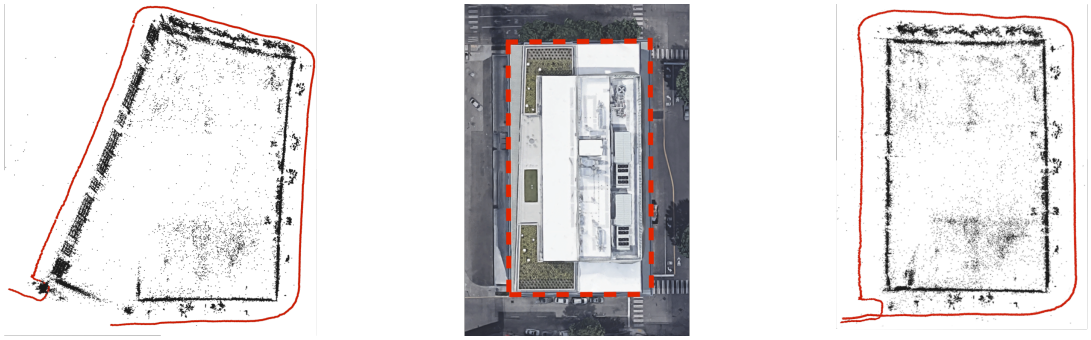
Periphery:	What does the world look like outside the camera's viewport?
Dis-occlusion:	What lies behind objects in the scene?
Geometry:	What is the shape and size of objects in the scene?
View-dependence:	How do object appearances change when viewed from other angles?

A large body of research is aimed at recovering spatial context from images, commonly referred to as *3D reconstruction*. These methods take as input a collection of images or videos and produce as output a three-dimensional (3D) model or implicit representation that can be used to answer the above questions. The recovered structural information can also be used in a number of different applications, such as rendering novel viewpoints in virtual reality, digitally extending the camera's field of view, or interactively modifying the scene by inserting objects or changing the lighting conditions.

3D reconstruction methods typically operate on multi-view data, *i.e.*, they reconstruct structural information by consolidating multiple observations of a scene from different viewpoints[1]. This consolidation process usually consists of a number of stages, including (but not limited to) camera calibration, localization, sparse reconstruction, depth estimation, fusion, meshing, and rendering. Unfortunately, each of these steps has associated limitations and failure modes, making the overall systems largely inaccessible to everyday users who may want to capture a three-dimensional copy of their environments but are unfamiliar with the idiosyncrasies of capturing a 3D scene. Additionally, these limitations make it particularly difficult to augment *prior* captures, in which the user was not explicitly intending to capture data for 3D reconstruction. In Chapter 3, we describe this general framework in detail, outline the common failure modes, and propose two novel extensions to the standard pipelines aimed at enabling 3D reconstruction in everyday, unconstrained settings.

These extensions focus on solving two main problems in 3D reconstruction: *drift* and *misaligned depth boundaries*. Drift, or the accumulation of small errors over time, is a

[1]or alternatively, use learned priors to estimate geometric information from a *single* image



(a) Sparse reconstruction **(b)** Satellite view **(c)** Our method

Figure 1.5: Reducing drift. Given a video of a building facade, we can use camera localization techniques to recover the camera trajectory (red) and a sparse point cloud (black) roughly delineating the structure of the building. Standard approaches for recovering this information suffer from drift (a), shown as bent or skewed structures that do not match the real scene, shown in (b), an annotated satellite image of the same building. Our method (c) automatically discovers and associates long-spanning structures (like an aligned row of windows or a flat wall) to reduce drift in the reconstructed poses, producing reconstructions that much more closely resemble the true scene.

common failure mode of camera localization — a first step in multi-view 3D reconstruction, where images (or frames in an input video) are localized in a consistent 3D coordinate frame. Drift typically appears in 3D reconstructions when using low field-of-view cameras (like those in smartphones) or when capturing very large scenes, such as entire buildings or cities. In Chapter 3.3, we propose a method to reduce (and sometimes eliminate) pose drift by leveraging long-spanning or repetitive structures in the scene (Figure 1.5).

Depth boundary misalignment is another common issue that occurs in dense depth estimation, a later stage of reconstruction where explicit geometric shape is estimated for each input image. Boundary misalignment occurs when the reconstructed geometry does not perfectly correspond with each object's boundaries in the input color image. As shown in Figure 1.6, this can result in reduced realism when using the reconstructed scene geometry for interactive applications, such as virtual object insertion or relighting. To solve this problem, in Chapter 3.4, we propose a *fast* smartphone-based dense depth estimation technique

(a) Input image **(b)** Virtual object insertion **(c)** Using our method

Figure 1.6: Misaligned depth boundaries. The estimated 3D geometry corresponding to a given input image (a) can be used for interactive applications, such as realistically inserting virtual objects into the scene (b) by using the geometry to decide when objects should be visible or occluded. Unfortunately, standard methods for 3D reconstruction produce imperfect or misaligned object boundaries, severely detracting from realism when synthesizing effects like these. Our method (c) solves this problem by explicitly optimizing for geometry suitable for these end-goal applications.

that produces geometry tailored for these types of interactive applications.

Overview We begin in the following chapter with a survey of modern forms of media, serving as inspiration for the different types of rich information that can be injected into older captured memories. In addition, we also review prior work in context recovery, and discuss the tradeoffs of different approaches. In Chapter 3, we discuss spatial context recovery, and present two novel approaches for recovering high-quality spatial and structural information from videos, with the goal of creating three-dimensional models that can be viewed in virtual reality or interacted with in augmented reality. In Chapter 4, we present methods for recovering temporal context, by injecting motion information to turn images into videos and videos into longer or looping videos. Finally, we conclude in Chapter 5 with directions for future investigation.

Chapter 2

BEYOND THE PHOTOGRAPH: A SURVEY OF MODERN MEDIA & AUGMENTATION TECHNIQUES

In this chapter, we provide a basic taxonomy of the types of visual experiences which have emerged over the recent decades, and focus in particular on those which may provide more information about the world than a static image. Additionally, we discuss methods for augmenting prior legacy captures, i.e., methods for converting older forms of media to newer, more immersive ones. To serve as a baseline for the forms of media presented in the remainder of the chapter, we begin in the next section by reviewing the visual and contextual limitations of the still photography camera.

2.1 The photograph

For over a hundred years, the still photograph has been the gold standard for capturing and sharing experiences. Even today, it serves as the predominant format for shared media on social media networks and digital media albums. Despite its popularity, and as briefly discussed in the introduction, still photographs have a number of significant limitations which inhibit their ability to faithfully reproduce the real world.

2.1.1 Analog photography

Historically, photographs have been captured onto analog material: film, paper, and even metal. This process of capturing light onto a physical substrate occurs through a chemical reaction: photographic film is made of a material that darkens or brightens when exposed to light. Eventually, this film is *developed* to prevent further reaction to light, preserving

Figure 2.1: Manual photo restoration. Analog photographs often contain artifacts, typically resulting from errors in the capture or development process (seen as white stains in the left photograph). Traditionally, these artifacts were corrected manually by a trained artist in a process called *spotting* (middle image). While this process could result in corrected photographs without noticeable artifacts (right), each photograph could take several hours to retouch.
Source: https://www.anatomyfilms.com/retouching-prints/

only the collection of light rays that were captured at a certain given moment. The film photograph (or more generally the analog photograph) has historically been associated with a number of artifacts caused by both (1) imperfections in the capture process and (2) film degradation over time, i.e., aging.

Analog photo restoration Removing these artifacts is not an easy task. For film photographs, this process was traditionally done by hand in a process known as *spotting*, where a trained artist painstakingly paints over artifacts in the printed photograph (Fig. 2.1). In the digital era, this same process can instead be performed digitally, by creating a digital copy of the analog photograph and restoring it through user-assisted editing tools such as Photoshop. While significantly easier and more accessible than their analog counterparts, these techniques still require manual labor and artistic expertise. To minimize human intervention, and thus further democratize the process of photo restoration, recent research has shown methods for performing restoration fully automatically, *i.e.*, without any human input [104, 4].

(a) Color inaccuracy **(b)** Shot noise **(c)** Rolling shutter

Figure 2.2: Photographic artifacts. Examples of some of the common visual artifacts typically seen in still photographs (above), and reference images without these artifacts (below).
lightstalking.com/restoring-old-photos/, photoreview.com.au/tips/shooting/how-to-control-image-noise/, youtube.com/watch?v=dNVtMmLlnoE

2.1.2 Color

Photographs also seldom accurately depict the colors of the world. This was especially true in the early years of photography, when cameras only captured black-and-white information, but even with the development of color cameras, there remained a significant gap between photographed colors and the corresponding colors in the real world. This phenomenon is attributed to the fact that photographic film is seldom sensitive to precisely the same wavelengths of light (*i.e.*, colors) as human eyes. To add to this, methods for printing or displaying photographs are often limited in their ability to accurately reproduce the depth of color and dynamic range that the human eye is capable of observing.

High-dynamic range imagery Newer capture technologies enable the acquisition of high dynamic range (HDR) images [31], which can capture scenes with significant differences in

intensity. While these images may more accurately represent the incident light rays arriving at the eye, viewing them still presents a challenge. Much progress has been made in recent years with HDR displays, but most digital displays and printing technologies are still unable to display differences in intensity as large as one may observe in the real world, e.g., a sunlit scene displayed on a digital monitor will not perfectly match the brightness or intensity of an actual outdoor scene.

Colorization In order to introduce color to early black-and-white photographs, photographers or artists would paint over printed photographs with colored paint. These days, this has been largely automated as a digital process, in which the hue or color saturation of different regions in the image can be adjusted semi-automatically in image editing software [85]. Additionally, newer techniques are able to perform this task fully automatically, by using priors about the colors of different objects learned from large collections of color photographs [197, 104]. While prior-based techniques present their own (ethical) challenges, discussed further in Chapter 5.1, these methods can often fully realistically colorize black-and-white or color-inaccurate photographs.

2.1.3 Digital cameras

With the advent of digital cameras, it became possible to bypass some of the imperfect chemical processes that converted light to film and film to paper. Digital cameras would instead capture light directly into digital buffers, which would not deteriorate over time, unlike their analog counterpart. Additionally, digital images could be displayed on digital monitors, precluding the need for development and printing. Despite all their benefits, however, digital cameras also introduced a number of their own limitations.

Nowadays, cameras are much more ubiquitous than they were several decades ago. This is largely a result of the inclusion of high-quality camera sensors in modern smartphones, a process that occurred through many years of optimizing the form factor of the digital camera. While this process has undoubtedly been a monumental success (given that smart-

phone cameras are widely used and capable of producing visually impressive photographs), the necessary optimizations for making cameras small and inexpensive enough to fit into consumer devices also introduced a number of artifacts. Among these was the introduction of the *rolling shutter*. Most modern smartphone camera sensors no longer capture the entire image at once, but rather (for efficiency) stagger the capture of each row of pixels in the image. This can result in undesirable image effects such as warping or stretching when the camera or scene is moving (Fig. 2.2c).

Rolling shutter correction Naturally, one may choose to capture an image with a global (*i.e.*, not rolling) shutter camera. However, in many cases, one may not be available, or the image may have been captured previously with a rolling shutter. In these cases, the effects of rolling shutter may be compensated for through careful camera calibration, 3D camera tracking, and scene motion estimation [8, 76, 52, 91, 121].

2.1.4 Noise

Photographic images also contain *noise* (Figure 2.2b), unwanted variations in intensity caused by a number of different factors (e.g. shot noise, hot pixels, banding). Digital cameras often reduce noise through careful engineering and post-processing. The amount of visible noise in an image can be reduced by adjusting parameters like the shutter speed, aperture size, and ISO (or gain). Newer digital cameras automatically (*i.e.*, algorithmically) adjust these parameters to maximize the image quality for a given lighting condition.

Burst photography When using a handheld camera, the necessary exposure time for satisfactory noise suppression may often result in an image with motion blur. As an alternative to increasing the exposure time, modern smartphone camera systems instead opt for capturing many images in rapid succession, each with a short exposure. As a post-process, these images can be spatially aligned and merged to produce a simulated long-exposure photograph with reduced noise [57, 47]. This same technique has been shown to work for a number of

Figure 2.3: Panoramas. A user can capture a panorama with a smartphone, by rotating the camera around a single point (left). The resulting panorama captures a full 360 degrees (right).

applications, such as high dynamic range photography [57], ultra low-light astrophotography [94], and superresolution [186].

2.1.5 Contextual limitations

So far, we have described visual artifacts, i.e., limitations that inhibit accurate representation and reproduction of *captured* information, but what about the information in the scene that is *not* captured? In many cases, the most important information in a scene may be just outside the camera's field-of-view, or may only be seen through continuous observation of the scene over time. This type of *contextual* information is often crucial to our understanding of a captured memory, describing where an image was captured or what actions were taking place, but is almost always missing from captured still photographs, since they only capture a fixed field-of-view and a single moment in time. In the remainder of this chapter, we focus on newer forms of media that capture more contextual information than a still photograph.

2.2 Panoramas and 360 degree imagery

Panoramic or *wide-format* photography is a technique for producing images that span wider fields-of-view than typical cameras (Figure 2.3). These images provide spatial context, extending the field of view beyond what a single image would usually capture.

Techniques for capturing panoramic photographs date back to the early years of photography. Originally, wider fields-of-view were achieved by capturing multiple images of different

viewpoints and physically laying out the resulting images side-by-side. Over the years, custom hardware was developed to bypass this process, such as panoramic cameras and special lenses, enabling one-shot capture of a wider field-of-view photographs. Nowadays, newer, portable versions of these cameras exist, such as the GoPro MAX and Ricoh Theta, that can even capture 360-degree imagery, i.e., incoming light from all incident directions.

Despite all the progress in developing newer hardware for panoramic capture, the most common approach for capturing a panoramic photograph is still to combine multiple images captured in different directions. However, unlike the early days of panoramic photography, modern techniques for merging these multiple images require significantly less manual intervention. These techniques typically require the user to perform a circular sweep, capturing a video of the surrounding scene. The resulting video frames are then automatically stitched together to form a seamless panorama (Figure 2.3) [168, 167, 17]. Effectively, these techniques enable all cameras to operate as wide field-of-view (or even 360-degree) cameras.

Simulated field-of-view While these techniques enable us to capture of wide field-of-view content without a wide field-of-view camera, they still require the user to perform a particular motion during capture. For cases in which the narrow field-of-view content is already-captured (i.e., we do not have the ability to capture more images), we can instead turn to a number of approaches that virtually *un-crop* images [148, 117]. These methods work by correlating structures seen in the input image with other images of the scene found online, and seamlessly stitching them together to produce a wide field-of-view image that preserves the source image's central content.

2.3 3D experiences

There is often much more spatial context within a scene than can be captured by a single (even panoramic) image. For instance, one may wonder what a particular object looks like from another viewpoint, or what the scene looks like behind a certain occluding object (i.e., beyond the capturing camera's line-of-sight). In order to answer these questions, we can

turn to methods that digitally reconstruct the geometric shape and appearance of the scene in three dimensions (3D), allowing us to render novel viewpoints that convey much of this missing information.

2.3.1 Capturing 3D via range imagery

At the time of capture, we can use a *depth* or *range* camera to capture detailed geometric information about the scene. In addition to capturing per-pixel color information, depth cameras also produce a per-pixel estimate for the physical distance of the camera from the observed object's surface. This depth information defines a 3D model of the scene, and can be trianglated into a meshed representation for convenient rendering. Additionally, depth information can integrated across multiple viewpoints (e.g., by capturing a video) to produce a more complete and physically accurate model [72, 30]. A number of different types of depth cameras exist, using technologies such as structured light and time-of-flight.

2.3.2 3D reconstruction from color images

In the absence of a depth camera, it is possible to recover geometric information from a collection of color-only images using *3D reconstruction* methods. These techniques typically operate by correlating textures and structures across multiple images of the same scene captured from different viewpoints [142, 43].

In cases where we only have access to a *single* color image of the scene, we can turn to single-view reconstruction techniques, which use the shading of objects [199] or learned priors about the shape of common objects [46, 138, 194, 46] to predict scene structure from a single image. While single view techniques are capable of producing very compelling results, they are typically less accurate than multi-view techniques, since multiple viewpoints are often necessary for resolving the many ambiguities of object shape and appearance, e.g., the bas-relief ambiguity.

In Chapter 3, I provide a more detailed overview of commonly-used multi-view 3D reconstruction pipelines and present two solutions for the common limitations encountered when

attempting to reconstruct 3D structure from in-the-wild smartphone footage.

2.3.3 Experiences

3D information enables reproduction of a scene's appearance from arbitrary viewpoints, but there are many ways in which this rendered information can be displayed. Similar to photographs, which can be viewed digitally or printed on paper, there exist multiple approaches for visualizing 3D content that differ in ease of use, accessibility, and immersion.

Two-dimensional rendering As a natural extension of the 2D photograph, we can simply display the novel viewpoints as two-dimensional images or videos. That is, we can render a virtual camera's viewpoint and display it as a static image, or we can generate a novel camera path throughout the virtual scene, and render the resulting frames as a two-dimensional video. Additionally, the virtual viewpoint can be interactively controlled by user input, such as with a mouse or touch-screen. This technique is popular for distributing 3D content and is employed in many commercially available solutions [150] since it does not require any custom hardware beyond a digital monitor.

Holography Recent advances have shown techniques for displaying captured 3D content as *holograms* [103, 75], i.e., semi-transparent three-dimensional images that exist within our physical space. These visual effects are typically created through selective diffraction of light, and can produce compelling 3D visualizations, but are typically limited in their visual realism due to the physical constraints of the display technologies.

Recently, similar technology has been employed in augmented reality glasses, such as the Magic Leap and Microsoft Hololens. Unlike prior techniques, which require a bounded space within which the hologram is displayed, augmented reality glasses enable the possibility of unbounded (i.e, infinitely large) virtual content.

Virtual reality Instead of displaying captured 3D models within our real world, we can instead *replace* the viewer's world with a virtual scene [59, 5, 3, 60, 102]. To do this, we can use virtual reality headsets, such as the Oculus Rift or HTC Vive, which are able to supply wide field-of-view imagery to both eyes independently at interactive frame-rates. By rendering the virtual scene twice (once for each eye, at their corresponding 3D positions), we can simulate the visual effect of binocular parallax, i.e., closer objects will be observed to have more disparity (between our two eyes). This effect, combined with the parallax resulting from head motion, can often convincingly reproduce the visual experience of moving around a real 3D scene.

These virtual reality experiences can range between (1) seated experiences, where the scene is intended to be observed with only head rotation and small, accidental head translation, and (2) room-scale experiences, where the viewer is able to walk around the scene and observe objects from widely different viewpoints. The choice between these two largely depends on the captured data available, i.e., if the quality of the reconstructed scene is sufficiently high to enable careful inspection from distant viewpoints. Additionally, the choice of 3D representation plays a large role in this decision, since certain representations do not support large changes in viewpoint.

2.3.4 Representations

Captured 3D content can be represented in many different ways. Each representation has its own set of benefits and drawbacks, and can facilitate different end-goal experiences.

Stereo pairs In many cases, if the user does not intend to vary their viewpoint, it is sufficient to provide only enough 3D information to produce the effect of binocular parallax, i.e., only enough information to display different images for the left and right eye. This is the case for 3D movies, in which two images are displayed simultaneously, and a pair of polarized or colored lenses display the corresponding image to each eye. In this case, however, the viewpoint of the camera is fixed, and does not correspond to the head motion.

A more immersive representation is the omni-directional stereo (ODS) format [129, 152], used in many capture setups like the Google Jump [3] camera system. The ODS format is effectively a 3D extension to the panorama, encoding the necessary light rays to enable rotational viewpoint changes, binocular parallax, and the subtle parallax that occurs due to head rotation.

Light fields Both of these representations are specialized forms of light fields [87, 48], the most comprehensive format for storing the appearance of a scene. Light fields store all the light rays that might be observed in a scene and parameterize each light ray by a position and direction, i.e. five degrees of freedom. To render a new viewpoint, one simply has to compute the parameters of the incident light rays to the virtual camera (using the novel viewpoint's position and orientation), and sample the corresponding intensities from the light field.

To create a light field of a real scene, it is necessary to capture a large number of images, such that all of the light rays required for reproduction have been captured in at least one input image. Light fields are, at least in theory, able to capture and reproduce a 3D scene identically. In practice, however, light fields seldom perfectly match real scenes, since they are either (1) not densely captured, due to the difficulty of capturing hundred or thousands of carefully placed images of an object, or (2) compressed or approximated, to avoid the excessive storage requirements of millions of light rays.

Textured meshes Instead of pre-determining the rays that can be rendered (and thus viewed), we can instead recover and store the geometric structure and appearance of the scene, such that we can render it from arbitrary viewpoints. The textured triangle-mesh is a standard choice for this, as it is commonly used in computer graphics and rendering pipelines, and is therefore efficient to store and render on a wide variety of devices.

Many of the above-defined 3D reconstruction techniques can be easily converted into a triangle mesh through well-studied approaches [78, 84]. To determine the surface appearance along the triangle mesh, we can apply standard techniques in projective texturing. For diffuse

objects, a texture map can be produced in a pre-processing stage by integrating all the color observations from different viewpoints. For more complex view-dependent appearances, the texture used on the mesh can vary as a function of the novel viewpoint [19, 62, 61, 126], enabling specular reflections and coarse transparency.

Volumes Some of the limitations of textured meshes include their difficulty in representing volumetric content, such as mist, fog, or clouds, as well as thin structures, such as power cables and strands of hair. Instead of representing 3D structure as discrete surfaces, we can use volumetric representations that represent a space as a collection of 3D points (or voxels) with varying color and opacity [110, 154, 99]. These voxels are then integrated along a particular viewing direction to generate the final observed color.

While these approaches can enable significant quality improvements over alternative techniques, generating and rendering volumetric content is generally considered to be less efficient than the alternatives, primarily due to the large number of points that need to be integrated per-pixel for high-quality rendering.

Layers In many cases, the appearance of a scene can be approximated sufficiently well by a series of layered planes (or other primitives) textured with a color and opacity map [145, 174, 5, 39, 130, 18, 185]. These representations are typically easy to reconstruct, since the associated geometric proxies are often both fixed and inexpensive to render. In these cases, reconstruction systems only need to estimate a series of color and opacity texture maps for each of the layers.

3D & Motion All of the above representations can also be temporally varying, i.e., representing more than a single moment in time. In general, introducing temporal variation vastly increases the storage complexity and typically requires carefully engineered rendering techniques [28, 18].

In addition to simply playing back a captured 3D scene, there also exist representations

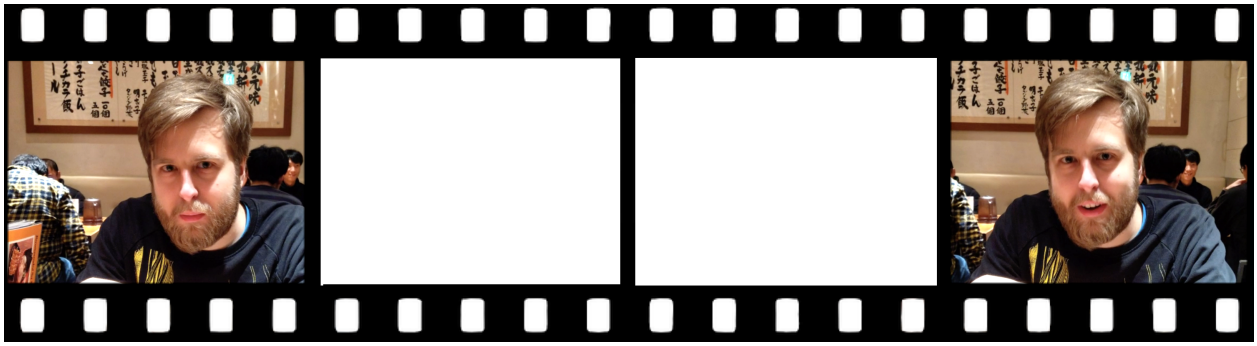


Figure 2.4: Static images lack context. In isolation, the first picture (left) may suggest the subject is unhappy, until a later frame of the video (right) clarifies that he was actually just caught mid-speech.

for interactively controlling the motion that occurred during capture, e.g., controllable re-animation of human subjects [127, 128, 99, 181].

2.4 Videos & burst imagery

All of the previously discussed forms of media focus on capturing and reproducing additional *spatial* context, but we have yet to see any that capture *temporal* context. As can be seen in Figure 2.4, temporal context is also very important in our understanding of captured memories, and can sometimes reveal a totally different meaning for a scene. Videos are a straightforward way of capturing temporal context, since they effectively capture multiple still images of a scene distributed over time. In contrast to still images, videos allow us to capture and reproduce many of the dynamic effects we observe in the real world, like object motion and changes in appearance.

Previously, videos we only able to be captured with dedicated video cameras, although nowadays the vast majority of available cameras (such as those in our smartphones) are also able to capture video content. Despite this, many of the experiences that would most benefit from the inclusion of temporal context (and thus would be best suited for video capture) are often still captured as static images. Whether this is due to convenience, preference, or habit

is uncertain, but the ultimate consequence is that many of our captured memories are much less immersive than they might have otherwise been. In this section, we discuss techniques for closing this gap.

Live photos Fortunately, if you use your smartphone to capture the image, this will very often also trigger the capture of a short video, or a *live photo*. These short stabilized videos are a byproduct of the fact that all captured pictures in mobile phones are created from a burst of frames captured just before the shutter was pressed (as described in Section 2.1.4). Unfortunately, these videos contain very few frames, and typically only span a second or so in time. As a result, they seldom capture particularly meaningful temporal information, and the captured motion is usually not particularly visually pleasing. In particular, if a live photo contains significant motion, this motion will be repeated (i.e., looped) every second or so, resulting in jarring or unpleasant playback.

Video textures, looping videos, and cinemagraphs This issue also exists in longer videos. Unlike images, which can be viewed continuously, videos always have an ending, at which point the viewer experiences a sudden transition to a frozen image, a blank canvas, or back to the start of the video. To make the visual experience more enjoyable, a lot of work has focused on exploring techniques for seamless video looping.

The terms *video texture* and *cinemagraph* are used often interchangeably to refer to seamlessly looping (or infinitely long) videos that do not contain a noticeable endpoint or looping point. Techniques that produce video textures or cinemagraphs typically take as input a longer video sequence and, through some analysis of the motion, produce a single seamlessly looping video, or an infinite (yet not obviously looping) video [140]. Traditionally, cinemagraphs were created by allowing a user to manually select certain parts of the frame for animation [6]. Newer approaches [171, 193, 93, 92, 119] perform this task fully automatically, determining which regions are easily looped, and which regions contain motions that are large in magnitude or otherwise unsuitable for looping. These approaches have also been

extended to operate on specific domains, such as videos of faces [6], urban environments [191], panoramas [2], and continuous particle effects [14, 95].

2.4.1 Animated images

In cases where our captured images are strictly static, we can instead *inject* temporal context, using techniques that perform image animation.

Recently, these techniques have gained popularity through commercial applications such as Plotagraph[1] and Pixaloop[2], which allow users to manually "paint" motion onto an image. The animations produced are not usually intended to simulate entirely realistic motion, but rather to inject life into an otherwise still photograph. These methods rely on some level of user interaction, typically to provide information about the direction, magnitude, and locality of scene motion. There are also ways of performing this annotation automatically, either through physical simulation, inherited motion from a source video, or various deep learning approaches.

Physical simulation Instead of manually annotating the direction and magnitude of motion, the motion of certain objects, such as boats rocking on the water, can be physically simulated, as long as each object's identity is known and its extent is precisely defined [26], or automatically identified through class-specific heuristics [73]. Since each object category is modeled independently, these methods do not easily extend to more general scene animation.

Exemplar-guided animation Alternatively, motion or appearance information can be transferred from a user-provided reference video, containing either similar scene composition [133], aligned information from a different domain, such as semantic labels [179], or unaligned samples from the same domain [151, 25]. Instead of a single user-provided video, a database of homogeneous videos can be used to inherit nearest-neighbor textural motion, assuming a

[1] https://app.plotaverse.com

[2] https://www.pixaloopapp.com

segmentation of the dynamic region is provided [120].

Latent space exploration. Recent advances in deep learning have enabled realistic, high-resolution image synthesis using generative adverserial networks (GANs). Many of these systems operate by representing images or scenes as a latent feature vector, which is decoded into a synthesized image. By perturbing the latent vector, or performing a randomized walk in the latent feature space, the resulting decoded images remain plausible, while also varying temporally [146, 63, 77]. These animations can visualize the space of possible appearances, but do not necessarily animate plausible motion.

Instead of a random walk, one can also directly control movement by applying spatial warps to latent features [98]. Still, deciding *how* to warp the image is non-trivial — to produce a realistic video, the applied transformations must correspond with feasible motion in the scene.

Using learned motion or appearance priors. Deep learning also enables motion synthesis from single-frame inputs [44, 177]. Similarly, video prediction methods [196, 190, 90, 189, 123] can predict future video frames from a single image, even modelling the inherent multi-modality of predicting the future. These techniques typically predict a set of future frames at once, and thus are limited to either low spatial resolution or few predicted frames.

In Chapter 4, we describe our approach to this problem, which takes as input a single image and automatically produces a high-resolution, seamlessly looping video with plausible scene motion.

2.4.2 Augmented-reality videos

With the increasing popularity of augmented reality experiences, we have seen a large number of ways in which videos can be enhanced with digital effects. These include virtual face masks that look as if they are attached to our faces, as well as virtual objects that can be placed in the scene around us. While these effects can often detract from the realism, this technology

allows viewers to *interact* with a captured video. This can enable more engaging experiences with our past videos, allowing us to redecorate an old apartment, improve the lighting in the video, or other artistic effects that magnify our past experiences. In Chapter 3.4, we present an approach that performs efficient and high-quality depth estimation to enable these types of interactions with previous captured smartphone videos.

Chapter 3

SPATIAL CONTEXT: TURNING IMAGES AND VIDEOS INTO INTERACTIVE 3D SCENES

3.1 Extracting 3D information from images or videos

As described in Chapter 2, many of the approaches that inject spatial context into images and videos do so by first extracting information about the 3D geometry of the scene. This 3D information can be reconstructed using a collection of well-studied methods typically referred to as *3D reconstruction* techniques. These methods take as input a set of images or a video and produce as output a 3D representation that can be used for arbitrary rendering or analysis, enabling a number of end-applications like (1) increasing the field of view, (2) rendering the scene from new viewpoints, (3) editing or relighting the scene, or even (4) adding content that realistically interacts with the scene. In this section, we describe one common pipeline for 3D reconstruction that can be used to achieve the above applications. In particular, we focus on a multi-view 3D reconstruction pipeline that uses classical techniques such as structure-from-motion and multi-view stereo. Note that while this is a very commonly used approach, many others exist that accomplish similar goals. For a more comprehensive list of different techniques for reconstructing and rendering 3D structure, we refer the interested reader to the in-depth surveys of Szeliski [166] and Hedman [58].

Capture Multi-view 3D reconstruction methods rely on the extraction of geometric information from *parallax*. In order for parallax to be observed, it is necessary for the same parts of the scene to have been seen from different camera positions. As such, when capturing data for multi-view 3D reconstruction, we typically opt for translational camera motion (as opposed to nodal pans, i.e., rotations), since this guarantees at least some amount of

parallax.

Camera pose estimation Given a set of captured images or video frames, the first stage of any 3D reconstruction pipeline is camera pose estimation, in which the 3D position and orientation of each camera is estimated in a consistent 3D space. Methods for camera pose estimation include structure-from-motion (SfM) and simultaneous localization and mapping (SLAM) systems. These systems typically take as input a collection of images or video frames, find matching structure across pairs of images, and use the parallax information extracted from these matching structures to jointly estimate camera motion and sparse scene structure. Additionally, if necessary, these systems can also jointly estimate camera parameters, such as focal length and lens distortion.

Dense depth estimation While the output of a pose estimation system produces an estimate for the structure of the scene, this estimate is typically very sparse, and does not fully represent the geometric shape of all objects in the scene. In order to estimate the remaining structure in the scene, we can turn to dense geometry estimation techniques, such as multi-view stereo. These techniques directly correlate photometric information (i.e., patches of pixels) across input images to determine a per-pixel estimate of scene goemetry. The output of a multi-view stereo algorithm is typically a depth estimate for each pixel in every input image.

Fusion In order to consolidate the depth estimates from all of the input images, we typically perform a process known as *fusion*, in which the depth points are projected into a consistent 3D space (using the previously estimated camera poses) and filtered for consistency. If our intended use for the estimated geometry requires explicit surfaces (i.e., not just 3D points), at this stage we can also perform meshing or other types of conversion to one of the representations described in Chapter 2.3.4.

Rendering Finally, once we have the camera positions and scene geometry estimated, we can render novel viewpoints or novel configurations through standard rendering pipelines. It is at this stage that we can adjust the parameters of the virtual camera to produce images or visualizations with additional context, e.g., by rendering the scene from new, unseen viewpoints, or increasing the camera's field of view.

3.2 Common issues

In the above pipeline, there exist a number of common failure cases. In this section, I provide a brief overview of these failure cases, and discuss some potential solutions.

Rotation-dominant camera motion As briefly mentioned above, camera pose estimation techniques typically require some amount of parallax in order to accurately estimate camera motion and scene structure. In cases where the camera motion is rotation-dominant (i.e., has very little positional translation) or the scene is too far away (and thus very large amounts of translational motion are required to observe parallax), the these methods can fail to produce correct camera pose estimates. This problem is still an active area of research, but a number of methods exist to remedy these failures, either by leveraging monocular depth information [80] or by making assumptions about the captured trajectory [164].

Drift Drift, or the accumulation of small errors over time, is a common problem in camera pose estimation. Since most pose estimation techniques rely on pairwise pose estimates between pairs of frames, longer translational sequences (e.g., when traversing around a large building) will chain these pairwise estimates together and accumulate any errors they may contain. This typically results in very skewed camera pose estimates, and can cause adverse effects further down the pipeline, hampering effective depth estimation. While there exist a number of techniques for reducing the likelihood and magnitude of drift, most rely on changes to capture hardware and methodology, such as the use of wide field-of-view camera, deliberate paths that increase pairwise redundancy, and maintaining sufficient distance from the

captured subjects. In the following section, we describe a novel approach for reducing drift after-the-fact, i.e., without any changes to captured hardware and without any additional instructions for the user.

Misaligned depth boundaries One common artifact in dense depth estimation is the misalignment of boundaries in the reconstructed per-frame depth maps and the corresponding object boundaries in the input color images. A related artifact is the failure to reconstruct geometric structure for thin structures that span only a few pixels in the captured images. These artifact both occur largely due to errors in earlier stages of the pipeline, such as poor calibration, poor pose estimates, or rolling shutter. Unfortunately, for many end-goal applications, the alignment of color and depth information is crucial in producing realistic renderings. A number of edge-aware image filtering approaches can be used to solve this problem, such as the bilateral filter, but these are typically costly operations, especially when used on large collections of images. In Section 3.4, we describe a method for accomplishing this goal efficiently by performing guided densification of sparse depth points (i.e., those from camera pose estimation).

View-dependent effects Objects with view-dependent effects, such as reflection and transparency, are difficult to reconstruct using standard multi-view photometric reconstruction, since these techniques rely on correlating object appearances across input images. When the appearance of an object changes across viewpoints, this correlation process becomes less reliable. The approaches that tackle this problem do so by explicitly modeling the changes of appearance as a function of the camera's viewing direction [110, 126]. This can enable both accurate scene reconstruction and more realistic scene re-rendering.

3.3 Resolving pose drift

Structure from Motion (SfM) [141] and Simultaneous Localization and Mapping (SLAM) [42, 37, 36] algorithms serve as the foundation for a wide variety of applications in computer

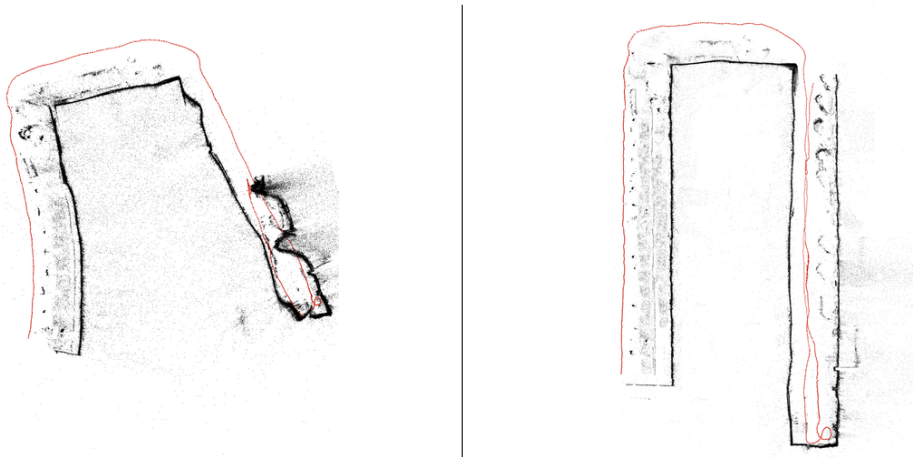


Figure 3.1: Drift. Standard SfM (left) suffers from significant rotation and scale drift. In scenes with long-spanning structures, like an aligned row of windows wrapping around a building, or a long planar surface, our method (right) can correct for this drift.

vision, including 3D reconstruction [1], 3D exploration of photo collections [156], and phone-based augmented reality [143]. Given a set of images as input, SfM and SLAM systems reconstruct the per-image camera locations and orientations, as well as a sparse set of 3D points.

Thanks to remarkable speed and accuracy improvements over the last decade [36], these camera-based tracking methods can be used to quickly reconstruct the layout or 3D model of a scene, simply by capturing a short video clip from multiple viewpoints. Creating an accurate model, however, requires highly accurate poses; but despite recent improvements, today's best algorithms still suffer from long-range *drift*, which results from the accumulation of small estimation errors caused by noise in feature point location estimates and other unmodelled sources of error. Significant drift can result in bent or deformed reconstructions, such as the one shown in Figure 3.1a.

While all reconstructions contain some amount of (sometimes unnoticeable) drift, certain camera configurations are significantly more susceptible. For instance, narrow field-of-view cameras, like those in mobile phones, are much more prone to drift, since features persist

for fewer frames when the camera is in motion. Since these feature tracks are the basis of motion estimation, if they are not observed from significantly different viewpoints, there are no direct constraints on the relative poses of those frames, and thus nothing to counteract the accumulation of drift error.

In fact, when trying to reconstruct objects which span a space larger than the field-of-view of the camera (such as a building or office-space), a drift-free reconstruction may be nearly impossible to capture. Those familiar with the limitations of these systems might attempt to choose camera paths that result in longer feature tracks, by keeping a greater distance from the subject, or fixating on single points; but often such paths are not feasible. For instance, to capture a building that is surrounded by trees, we're forced to capture the building from a very small distance. In such cases, our best option is to face the object and trace its perimeter, but doing so can result in significant drift (Fig. 3.1a).

In this section, we demonstrate how tracking *extended virtual features* such as co-planar sets of points, which span non-overlapping image frames or sometimes even complete video sequences, can dramatically reduce accumulated drift, enabling us to reconstruct sequences such as those described above. We focus on extended features which are *structural*, namely vanishing points and oriented planes, which are common in man-made environments, such as cities and buildings. Note that our system does *not* make Manhattan World assumptions, i.e., that all dominant structural planes are orthogonal. It can handle Atlanta World scenes with structures at arbitrary orientations [139].

To demonstrate the effectiveness of such features, we incorporate them into Theia [164], a popular open-source SfM system, and evaluate on both synthetic and real-world captures of scenes with strong planar structures using low field-of-view cameras. We show that the addition of extended features drastically reduces drift in both cases, producing results that surpass in quality both Theia and COLMAP, two of the leading open-source SfM packages.

We show the utility of extended features applied to a global SfM framework. Global SfM systems, in contrast to *incremental* systems, solve for all camera poses at once, and are typically much faster, albeit less robust to errors in pairwise pose and correspondence. We

show that for our drift-prone sequences, adding extended features to a global SfM system can produce higher quality reconstructions than an incremental SfM system at a fraction of the processing time. This is possible because our modifications preserve the efficiency of previous global methods, only adding a small number of extra parameters and a linear number (in the number of cameras) of additional constraints.

We begin in Sec. 3.3.1 with a review of the previous literature, followed in Sec. 3.3.2 with a description of our global SfM baseline. Sec. 3.3.4 then describes the structural components of our algorithm, including the use of vanishing points for orientation estimation, local orthogonal plane fitting, and the integration of these planes into the global position solver. Sec. 3.3.5 presents our experimental results on both synthetic data and real-world mobile phone captures. We then wrap up with a discussion of our results.

3.3.1 Previous work

SfM and SLAM methods generally consist of the same set of components: identifying correspondences, pairwise pose estimation, and global integration.

Correspondence. Both SfM and SLAM methods begin by establishing correspondences, which capture the motion observed between pairs of frames and serve as the foundation for pose estimation. Traditionally, the features used for correspondence are 2D point descriptors [54, 100, 175]. These descriptors analyze local image patches to determine salient keypoints and are invariant to geometric and radiometric transformations. Other types of features include line segments [7, 137, 109], vanishing points [153], 3D planes [32, 83, 137, 13, 12], and inertial measurements [89].

Pairwise pose estimation. Once the correspondences have been established and filtered, two-view or multi-view geometry estimation techniques [116, 132, 53, 38, 55, 56, 172] are used to recover relative camera pose between pairs of images, which, along with point correspondences, can be used to triangulate 3D points. These pairwise relative camera poses and 3D points are effectively *local* reconstructions consisting of two or three frames.

Global integration. In order to produce a global reconstruction, it is necessary to combine the local pose estimates. Techniques for robustly integrating these local reconstructions are usually divided into two categories: incremental and global.

Incremental methods [155, 187, 141, 1, 41, 164] have long dominated the state of the art, and can be found in the majority of open-source implementations, such as Bundler [155], VisualSfM [187], and COLMAP [141]. This form of global integration typically starts with a carefully selected two or three-frame reconstruction [156, 155, 141], and incrementally grows it by selecting and anchoring new frames to the already-reconstructed cameras and points. To account for the inconsistencies from the newly registered poses, each frame addition is typically followed by a number of filtering operations to verify and refine the pose [156, 141]. While these filtering operations cause the method to be very robust to outliers, they can also be very computationally expensive. To improve efficiency, incremental systems employ a number of optimizations to reduce incurred processing time [15, 157, 1, 188, 114]. SLAM methods [37, 112, 36, 134, 40] also belong to this category, as they incrementally track the camera pose by accumulating differential camera motion.

Global methods [49, 137, 153, 74, 111, 184, 109, 163] estimate all camera poses simultaneously, making them efficient for large-scale problems. These methods are generally regarded as being less robust to outliers, as the lack of an incremental reconstruction precludes the ability to identify pairwise pose outliers by verifying against a global model. To improve robustness, global SfM methods utilize either groups of frames [111, 74, 50], observed 3D landmarks [153, 29], or pairwise geometry analysis [157] to further verify pairwise pose estimates before integrating globally.

Drift Mitigation. Incremental and global techniques both suffer from drift, particularly in long sequences without loop closures, caused by the accumulation of small relative pose errors. A number of avenues have been explored to mitigate these errors.

SfM systems will often perform global bundle adjustment [173] over all reconstructed frames and points. This process can reduce, *but not eliminate*, drift error, since the bundle-adjusted reconstruction can only be as good as the correspondences used for optimization.

If there are correspondence errors, the resulting reconstruction will inevitably contain some amount of drift.

Real-time SLAM methods [37, 112, 36, 134, 40] which cannot afford costly global bundle adjustment will often include inertial measurements [89] as a secondary source of motion information. While these inertial sensors have become commonplace in modern mobile phones, most captured or distributed videos do not retain inertial measurements, and thus this information is often unavailable.

For closed-loop sequences, a common tactic to mitigate drift is to perform *loop closure* or explicit matching and pairwise pose estimation between temporally distant frames that observe similar parts of the scene. While the constraints derived from these matches have the potential of reducing the drift in the reconstruction, in many cases, the drift error is simply redistributed to different parts of the reconstruction. This is because loop closure only adds constraints to a handful of images at the closure point, but does not apply direct constraints to the poses of images elsewhere.

Higher-dimensional geometric features, like vanishing points, have been shown to significantly reduce rotational drift in both SLAM [21] and SfM [153], since they are able to extract direct constraints between pairs of frames that do not observe the same part of the scene. For active depth sensors, dense normal statistics [159] provide a similar signal. These methods all provide strong constraints on rotational drift, but do not address *translational* drift.

Other geometric features, such as 3D lines, have shown potential for reducing translational drift. Micusik *et al.* [109] incorporate line segment endpoints into an incremental SfM system, allowing accurate reconstruction in the absence of dense point features. Similarly, Zhou et al. [201] propose a SLAM system that extracts and matches axis-aligned structural lines to apply constraints on the camera pose. Nurutdinova et al. [118] propose a generalization that includes arbitrary 3D curves in bundle adjustment. These methods rely on establishing correspondences between lines, either through endpoint matching or photometric comparison, techniques which, in real-world settings, are typically less reliable than point-based features.

Planar constraints have also been shown to resolve translational and scale drift. Szeliski and Torr [169] provide a theoretical introduction to the use of known planar structure as constraints in bundle adjustment. They show that for simple dataset of two cameras, the quality of reconstruction can be improved significantly by incorporating prior knowledge of planar scene structure into bundle adjustment. Extending upon this work, Rother [137] proposed a factorization-based reconstruction system which jointly reconstructs camera pose, points, lines, and planes. Similar to [169], results are only shown on small scenes with few images, since factorization-based approaches are typically quite sensitive to outliers, and do not easily scale to large real-world datasets. Additionally, both methods require a known reference plane, unlike our method, which automatically discovers and associates structural elements.

More recently, Liu et al. [97] demonstrated a SLAM framework that applies a piecewise-planar assumption in tracking, using homographies to efficiently track a camera under rapid motion. While the use of homographies enables fast and robust tracking, the method still suffers from significant drift over longer sequences, as planes are not associated across non-overlapping views.

Li et al. [88] show that additional constraints, such as coplanar sets of lines, found through vanishing point estimation and homography fitting, can further reduce positional drift. This method is reliant on long-spanning line segments in order to establish strong constraints, and requires mutually visible line endpoints for optimization. In contrast, our method does not require lines to be mutually visible in multiple views, and only relies on feature point correspondences, which are standard for SfM systems. Yang and Scherer [192] show that the boundaries between the semantic labels of surfaces such as walls, floor, and ceiling can also be used in constraining the camera position. This method relies on automatic semantic labeling, and thus does not easily extend to arbitrary planar structures. Cohen et al. [27] show that many of these same assumptions, such as Manhattan-oriented structure, symmetry, and repeating elements, can facilitate the fusion of disconnected or sparsely overlapping reconstructions.

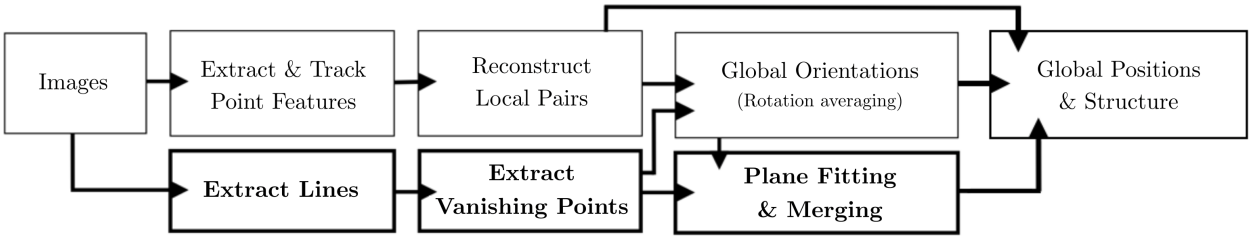


Figure 3.2: Overview. An overview of the global SfM system, showing the baseline components (non-bold) and our added structural constraints (bold).

In this section, we describe a technique [65] that automatically extracts structural elements from a real-world video sequence, including vanishing points and planes, and automatically finds associations between non-overlapping observations of these elements in order to establish long-range constraints that reduce pose drift. Most similar to our work, Shariati et al. [149] jointly optimize for wall positions relative to the camera, but require depth and inertial sensors, and also assume Manhattan structure. Our method takes as input only a monocular video sequence, and easily extends to scenes without a global Manhattan coordinate frame.

3.3.2 Baseline global SfM pipeline

This section describes the baseline Theia system [164], shown as non-bolded boxes in Fig. 4.1. Section 3.3.4 describes our novel structural constraints, which are shown in bold. Theia is a point-based SfM system, using tracked point features to create local reconstructions, which establish consistent relative camera motion between pairs of frames. These local reconstructions are then used in rotation averaging [23] to estimate the global orientations of all the cameras. Finally, the global camera centers are computed using the estimated rotations, and the 3D points are triangulated to produce the final reconstruction.

Theia implements a number of global rotation and position solvers. For our baseline system, we chose the L1-IRLS rotation solver [23] and the LUD position solver [122]. These are the recommended default solvers in Theia and also consistently produced the best results

on our datasets. Apart from two modifications that aid in handling degenerate configurations, the implementation of Theia is unchanged from the publicly available library. Here, we provide a quick review of the formulation used in both the rotation and position solvers.

Problem formulation and notation We start with a set of *point tracks* relating 2D point features $\mathbf{x}_{ip}$ with camera (image) indices i and 3D point indices p. Since we assume known camera intrinsics, these points have already been centered w.r.t. the image center and their pixel coordinates divided by each camera's focal length f_i so that the $\hat{\mathbf{x}}_{ip} = (x_{ip}, y_{ip}, 1)$ correspond to metric (Euclidean) ray directions in each camera's frame.

The projection equations relating a 3D world coordinate $\mathbf{p}_p$ to its corresponding 2D projection $\mathbf{x}_{ip}$ in image i is then

$$\hat{\mathbf{x}}_{ip} \sim \mathbf{R}_i(\mathbf{p}_p - \mathbf{c}_i), \tag{3.1}$$

where $\mathbf{R}_i$ is the camera's 3D orientation, $\mathbf{c}_i$ is its 3D position, and $\sim$ indicates similarity up to scale.

Global orientations (rotation averaging) To solve for the global orientations of all cameras, global SfM systems typically use rotation averaging, which solves for the global camera orientations that best agree with pairwise rotation estimates by minimizing the consistency error

$$E_{\text{rot}}(\{\mathbf{R}\}) = \sum_{i,j} ||\mathbf{R}_j^{-1}\mathbf{R}_{ij}\mathbf{R}_i - \mathbf{I}||_1 \tag{3.2}$$

where $\mathbf{R}_i$, $\mathbf{R}_j$ are the unknown global orientations of frames i and j, respectively, and $\mathbf{R}_{ij}$ is the known pairwise rotation estimate between the two frames.

We use Theia's robust L1-IRLS solver, proposed in [23], which first performs an L1 minimization and then refines the solution using iteratively reweighted least squares.

Global position estimation Once the global orientations of all cameras have been estimated, all pairwise constraints are then integrated to estimate global camera centers. Theia uses the "least unsquared deviation" (LUD) global position estimator proposed by [122],

which formulates the optimization as:

$$E_{\text{pos}}(\{\mathbf{c}, s\}) = \sum_{i,j} ||s_{ij}\mathbf{t}_{ij} - \mathbf{c}_j + \mathbf{c}_i||_2 \tag{3.3}$$

where $\mathbf{t}_{ij}$ is a known pairwise translation estimate (after rotating all cameras into the global coordinate system), $\mathbf{c}_i$ and $\mathbf{c}_j$ are the unknown camera centers, and s_{ij} is the unknown scaling coefficient for the pairwise reconstruction. These equations are optimized using a fast convex solver.

3.3.3 Extensions to handle degenerate configurations

In order for the baseline Theia to work on our sequences (narrow field-of-view videos), which often contain "degenerate cases" such as linear camera motions and single planes (building walls), we added the following extensions, which are described in more detail in our supplementary material:

1. The above global position estimation suffers from degeneracy for colinear camera motion (as described in [74, 111]). In order to resolve this, we adapt a simplified version of [111], applying additional constraints on the relative scales of pairwise reconstructions by comparing the triangulated depths of shared feature tracks.
2. In order to deal with planar scenes and pure rotations, for which the 5-point algorithm may produce degenerate configurations, we additionally estimate pairwise pose from a homography (using [106]), keeping whichever approach produces a larger number of inliers.

Co-linear camera motion Pairwise constraints are insufficient for certain camera configurations, such as co-linear motion, which is often found in video-based reconstruction. A common tactic for reducing ambiguity and error in the final reconstruction is to verify pairwise estimates among local groups of three frames (triplets) [74, 111]. These methods triangulate local 3D points between every pair of the cameras within a triplet and then compare the point depths to establish scale constraints between pairwise translation estimates.

In order to make Theia robust to co-linear motion, we use a variant of [111], establishing

scale constraints between pairs of local pairwise reconstructions and integrating them into Theia's robust position solver [122].

In detail, we first search for triplets as any three frames that have valid three-way pairwise pose estimates. Local 3D points are triangulated for each pairwise estimate in the triplet, using the relative rotations and translations they define. Then, relative scales are computed between pairs of these pairwise estimates as the robust ratio between the triangulated point inverse depths. These scale constraints are added as soft constraints to the previous linear system (Equation 3.3) as:

$$E_{\text{scale}}(\{s_{ij}, s_{kl}\}) = W_s(ij, kl) * (r_{ij \to kl} s_{ij} - s_{kl}) \,, \tag{3.4}$$

where s_{ij}, s_{kl} are the global unknown scale values for pairwise local reconstructions $i \to j$ and $k \to l$, and $r_{ij \to kl}$ is the previously computed robust ratio. The strength of the added constraint is weighted by $W_s(ij, kl)$, defined as

$$W_s(ij, kl) = \max\left(\frac{N_{\text{pts}}(ij, jk)}{N_{max}}, 1\right) \tag{3.5}$$

where $N_{\text{pts}}(ij, jk)$ is the number of shared tracks with valid triangulations[1] between pairwise reconstructions $i \to j$ and $j \to k$, and N_{max} is the number of points at which the weight saturates (we use $N_{max} = 500$).

Rotation-only camera motion and planar scenes At the core of Theia's global SfM pipeline lies its pairwise relative pose estimates, which are used as constraints in both in the global rotation and position solvers. These pairwise relative poses are estimated using the five-point algorithm [158, 116], which is known to produce unreliable estimates in cases of purely rotational motion or entirely planar scenes. Since our sequences largely consist of close-up captures of building facades (planes), we incorporate a secondary pipeline for pairwise relative pose estimation.

[1] as defined by Theia's standard two-view triangulation code, which includes triangulation angle, among other metrics

In addition to the usual five-point estimation, we estimate a homography between the matching feature points. If it is determined that a majority of points are considered inliers to the homography, the estimated homography is decomposed using [106], resulting in four candidate rotation-translation transformation pairs. These putative transformations are subsequently filtered by first discarding those which triangulate points with mostly negative depths, and then the transformation with the smallest reprojection error is retained. The remainder of the pose estimation pipeline remains unchanged.

3.3.4 Structural constraints

In order to reduce the drift (global low-frequency errors and deformations) in our reconstructions, we exploit large-scale *structural constraints* such as vanishing points and planes. These can be thought of as *extended features* since they will often span many more frames than traditional point tracks, which come in and out of view.

Lines and vanishing points

In order to obtain a drift-free set of rotation estimates, we first compute for each frame (wherever possible) a vertical vanishing point and one or more horizontal vanishing points, as described in the supplemental material and illustrated in Fig. 3.3. Once the vanishing points have been found in frame i, we estimate a global rotation $\mathbf{R}_i^{\mathrm{vp}}$, which maps one of the Atlanta world horizontal directions to the dominant horizontal direction in the frame. Compared to the previous work by Sinha *et al.* [153], which performs a global matching step between vanishing directions of all frames, our approach is tailored for continuous video sequences where motion between frames is small, so the vanishing point associations can be chained through consecutive frames. This gives us a consistent *soft* (drift-free, but empirically lower-accuracy) global orientation constraint for every frame that has associated vanishing point data.

To take advantage of these estimated points, we initialize the solution for the global camera orientations using the vanishing points, $\mathbf{R}_i \leftarrow \mathbf{R}_i^{\mathrm{vp}}$. In addition to the inter-frame

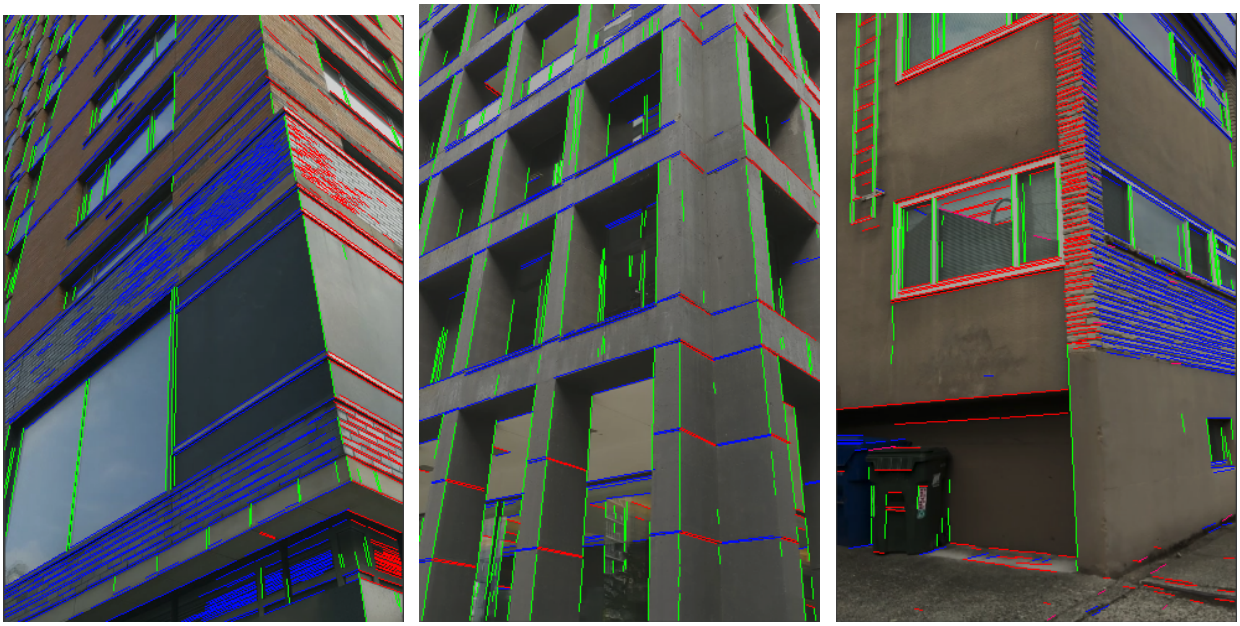


Figure 3.3: Vanishing points. Examples of our edge detection and vanishing point estimation stage. Different colored edges indicate their membership in different vanishing points. Note that only two orthogonal vanishing points need to be detected to establish the coordinate frame.

constraints from pairwise pose estimates (3.2), we also add penalties on the difference between the current rotation estimates and the vanishing-point based rotations:

$$E_{\text{vp}}(\{\mathbf{R}\}) = \lambda_{\text{vp}} \sum_i W_{\text{vp}}(i) \|\mathbf{R}_i^{-1}\mathbf{R}_i^{\text{vp}} - \mathbf{I}\|_1. \tag{3.6}$$

These constraints are weighted using a regularization parameter λ_{vp} and a per-frame weighting function

$$W_{\text{vp}}(i) = \text{clamp}(1 - \frac{\Delta\theta}{\theta_{max}}, 0, 1) \tag{3.7}$$

where $\Delta\theta$ is the incremental rotation between frames $i-1$ and i and θ_{max} is the largest tolerated incremental rotation. We use $\theta_{max} = 5$ degrees for all experiments. This weighting function makes the rotation estimates more robust to significant outliers in the vanishing point estimates, by lowering the weight of consecutive frames when they are very different. Further information on vanishing point estimation and integration is provided in the supplementary material.

Vanishing point estimation and integration In order to obtain a drift-free set of rotation estimates, we first compute for each frame (wherever possible) a vertical vanishing point and one or more horizontal vanishing points. We first detect line segments using the LSD line segment detector [51]. To fit vanishing points, we use the general expectation maximization approach of [81]. In the expectation stage, line segments are softly associated to vanishing points; in the maximization stage, vanishing points are fit to line segments by solving a weighted least squares problem on the unit sphere. Between iterations, we merge vanishing points that become sufficiently close, and purge vanishing points with low evidence.

This approach requires a method to find an initial set of vanishing points. For this, we start by finding vertical vanishing point candidates from lines that are close to vertical in the image. For each candidate vertical direction, we compute the corresponding horizon line. Candidate horizontal vanishing points are found by intersecting image lines with the horizon line. We divide the horizon line into equal-angle bins and select candidate vanishing points from peaks in the resulting histogram. We can also obtain a candidate set of vanishing points from the previous frame, if any. To select among the candidate sets, we choose the set which maximizes the total length of lines associated with a vanishing point.

Once the vanishing points have been found in frame i, we want to estimate a global rotation $\mathbf{R}_i^{\text{vp}}$, which maps one of the Atlanta world horizontal directions to the dominant horizontal direction in the frame. As mentioned earlier, our method is tailored for continuous video sequences where motion between frames is small, so the vanishing point associations can be chained through consecutive frames. For most cases, the vanishing directions are consistent across consecutive frames, since the distribution of edge orientations in the image remains approximately constant, and thus the orientation can be computed as the relative rotation from the dominant horizontal axis. However, in certain cases, for example when turning a corner, the dominant vanishing direction may change. In these cases, we use the pairwise pose estimate from the last frame with valid vanishing directions to verify the association. This verification can result in either keeping the estimated vanishing point as-is, applying a 90 degree rotation (eg. turning the corner of a building), or begin tracking a

new set of horizontal vanishing directions altogether (eg. when turning a non-orthogonal building corner, like in an Atlanta-world). We keep whichever association produces the most consistent vanishing point association, and invalidate any associations with errors greater than 10 degrees. While this same verification process could be used to associate vanishing points in arbitrary frames, we still limit our method to sequential data, since we found empirically that lifting this assumption, i.e. performing this association between all frames with pairwise estimates, often results in spurious VP associations.

Once the local coordinate frames have been estimated, they need to be integrated into the rotation averaging step. These constraints are weighted using a regularization parameter λ_{vp} and a per-frame vanishing point weighting function W_{vp}, which we define as

$$W_{\text{vp}}(i) = \max(1, 1 - \frac{\Delta\theta}{\theta_{max}}) \tag{3.8}$$

where $\Delta\theta$ is the incremental rotation between frames $i-1$ and i and θ_{max} is the largest tolerated incremental rotation. We use $\theta_{max} = 5$ degrees for all experiments. This weighting function makes the rotation estimates more robust to significant outliers in the vanishing point estimates, by lowering the weight of consecutive frames when they are very different.

In order to determine a suitable value for λ_{vp}, we could estimate the variance of the local inter-frame rotation estimates by comparing them to the rotation-averaged solution, and the variance in the vanishing point estimates by comparing incremental rotations between them to the rotation averaged differences. Instead, we took the simpler approach of just setting a value empirically by observing the rotation averaged plots for various values of λ_{vp}. Fig. 3.4 shows the orientation estimates for a representative set of frames (the last 322 frames of the `MORE_HALF` sequence) reconstructed using various values of λ_{vp}. Notice that with weak vanishing point constraints ($\lambda_{\text{vp}} = 0$ or 0.01), the results drift significantly from the global orientations given by the vanishing points. On the other hand, the vanishing point estimates are occasionally wrong, such as the large spike seen around frame 1075. We found experimentally that setting $\lambda_{\text{vp}} = 10$ gave us good results for all of the sequences that we tested. Using this value, we show in Fig. 3.5 that adding vanishing point constraints to

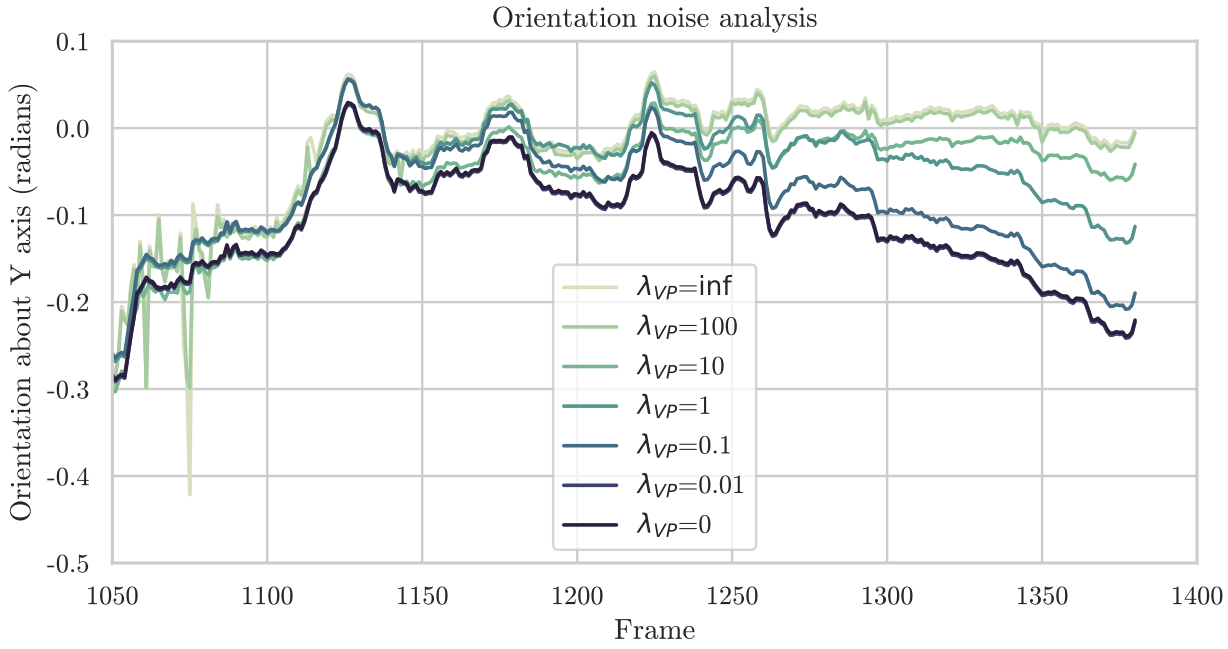


Figure 3.4: Effect of vanishing point constraints. A comparison of the reconstructed camera orientations for the `MORE_HALF` sequence using different values of λ_{vp}. We see the reconstructed orientations suffer from significant low-frequency drift when no vanishing point constraints are applied ($\lambda_{vp} = 0$). This is equivalent to result shown in Figure 8a of the main paper. Increasing the weight of the vanishing point constraints causes the low frequency drift to decrease ($\lambda = 0.1, 1, 10$), but too large values ($\lambda = 100$) will introduce high-frequency noise from the vanishing point estimates into the reconstructed orientations, seen as spikes in the curve.

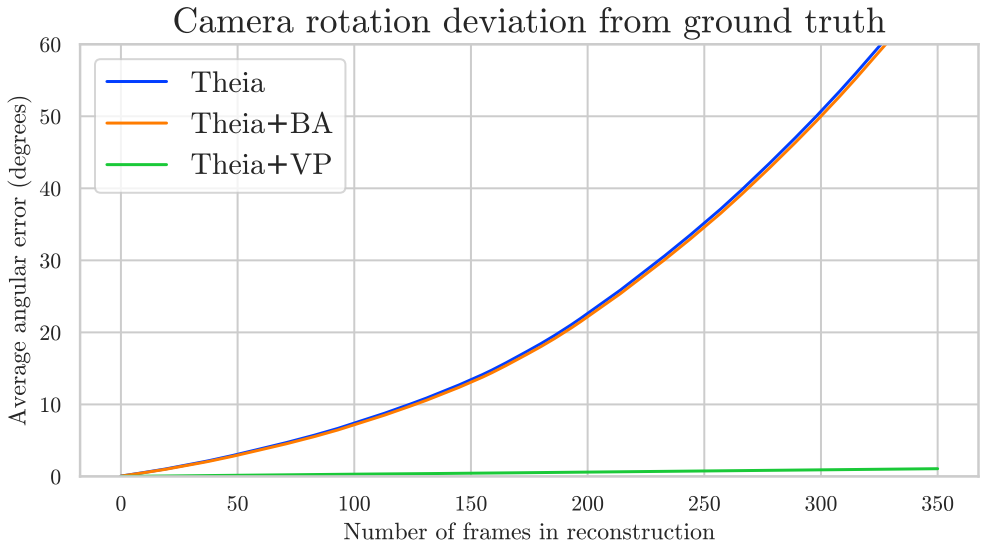


Figure 3.5: Quantitative evaluation (rotational drift). A comparison of rotational drift using different combinations of Theia with global bundle adjustment (BA) and added vanishing point constraints (VP). We see that the added constraints practically remove the observed rotational drift, and without the added constraints, even bundle adjustment has difficulty converging on the correct solution. In order to be robust to pose errors in individual frames, we align the reconstructions to ground truth by estimating a similarity transformation.

rotation averaging results in the elimination of rotational drift in our synthetic sequence.

Extended planes While adding vanishing points as soft constraints on camera orientations can dramatically reduce orientation drift and increase reconstruction accuracy, positional drift still remains an issue. In order to address this, we exploit another major source of structural constraints, i.e., coplanar points arising from man-made structures such as buildings.

In this section, we describe how we identify coplanar 3D points in local pairwise reconstructions and then link these together into extended global planes that can be used as additional constraints in the pose estimation process. As mentioned before, in our current system, we restrict our attention to planes whose normals correspond to one of the dominant vanishing point directions.

Local plane fitting.

We begin by discovering planes in each pairwise reconstruction. For each local reconstruction containing valid vanishing points, we use the pairwise pose estimate to perform two-view triangulation, resulting in a local 3D point cloud. We then perform a plane sweep along the three orthogonal vanishing directions associated with the base frame of the pairwise reconstruction. This results in a number of local planes π, parameterized as:

$$\mathbf{p} \cdot \hat{\mathbf{n}}_{\pi}^{ij} = d_{ij}^{\pi}, \tag{3.9}$$

where the $\mathbf{p}$ are the local 3D point inliers, and $\hat{\mathbf{n}}_{\pi}^{ij}$ is the local plane normal, and d_{ij}^{π} is the distance along the normal $\hat{\mathbf{n}}$ from the origin of pairwise estimate $i \rightarrow j$.

Here, we provide a more detailed description of the plane sweep process. We begin by projecting the 3D points along the direction we plan to sweep (i.e. performing a dot product of the direction vector and the 3D point). The resulting projected points are then sorted by depth (distance along the sweep direction from the pairwise origin) producing a one-dimensional histogram. A moving window algorithm is used to detect peaks in this histogram. Peaks correspond to sets of coplanar points along the sweep direction. In peak detection, we

use a window size proportional to the predicted scale of the pairwise reconstruction and also weight points by their inverse depths, since closer planes provide more useful constraints. We only retain peaks corresponding to at least N_{min} points (in practice, we use N_{min}=3). Finally, each peak is used to parameterize a local plane π, with $\hat{\mathbf{n}}_{\pi}^{ij}$ equal to the sweep direction, and d_{ij}^{π} equal to the depth of the peak.

Plane merging and constraints. Once local plane hypotheses have been generated for each pairwise reconstruction, we group these into global *extended* planes, which we then use to provide additional constraints on the local scales and global camera positions, as described below in Eq. 3.10. In order to link these local hypotheses, we rely on point correspondences. Since local planes are established by finding co-planar sets of 3D points, each local plane contains a set of *inlier* tracks which can be used to associate local planes with one another. We perform this association by greedily merging any two local planes which share a majority of tracks, and are associated with the same vanishing direction.

Once we have established estimates of which local plane hypotheses correspond to one another, we define constraints that encourage these local planes to coincide in the final 3D reconstruction. In order to integrate local plane hypotheses into global constraints on camera positions, we add scalar variables d_p to the linear system, where p is the index of the global extended plane (as opposed to the local plane index π). These variables define each global plane's location (distance to the world origin along the plane normal). This corresponds to one added constraint for each observation of a global plane p in a pairwise estimate ij:

$$E_{\text{pln}}(\{\mathbf{c}, s\}) = \sum_{i,j,p} W_{\text{p}}(ij, \pi) \| s_{ij} d_{ij}^{\pi} + \mathbf{c}_i \cdot \mathbf{n}_p - d_p \|_2 \tag{3.10}$$

where the s_{ij} is the unknown pairwise estimate scaling factor, $\mathbf{n}_p$ is the known plane orientation (normal vector), $\mathbf{c}_i$ is the unknown global camera position of the base camera in the pairwise transformation, and d_p is the unknown global distance of the plane from the world origin along $\mathbf{n}_p$. Fig. 3.6 provides a visualization of these terms.

The number of global planes p (and hence extra scalar unknowns) is a small constant number per scene, and therefore the number of added constraints to the system is linear in

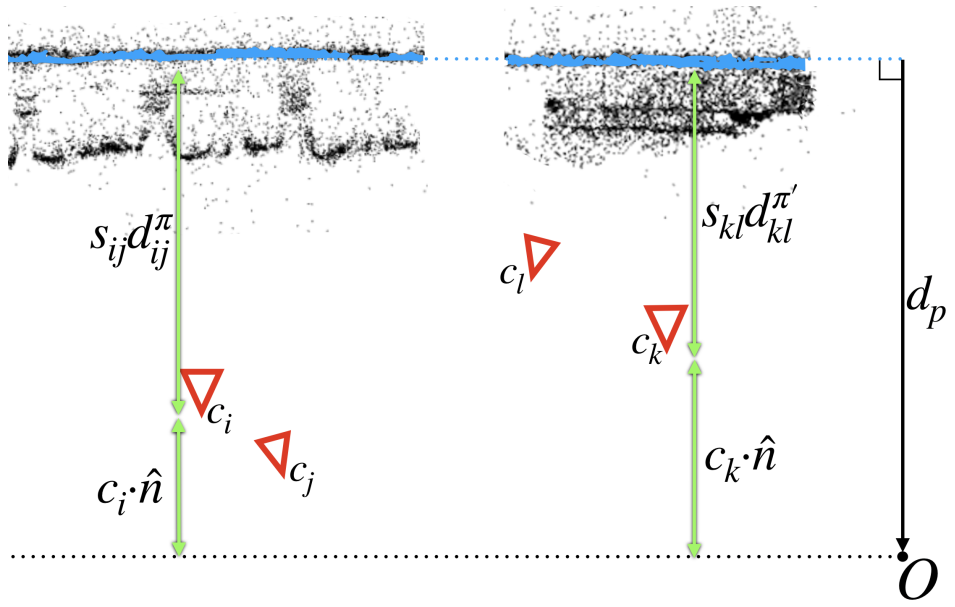


Figure 3.6: Planar constraints. An illustrated example of our plane constraints described in Eq. 3.10. In the global coordinate frame, two pairwise local reconstructions $i \to j$ and $k \to l$ have their own local distances d_{ij}^{π} and $d_{kl}^{\pi'}$ between the base cameras and their respective local planes π and π'. When these planes correspond to the same global plane p (blue), our plane constraints will align these two locally estimated planes in the global reconstruction by encouraging each local plane distance to be consistent with a single global distance from the origin d_p. This example shows the result of optimization, where our planar loss has been minimized such that $d_p = s_{kl}d_{kl}^{\pi'} + c_k\hat{n} = s_{ij}d_{ij}^{\pi} + c_i\hat{n}$, and $E_{\text{pln}} = 0$.

the number of cameras. We weigh each of these constraints by the support of the global plane in the local pairwise reconstruction, i.e. by the number of inlier tracks. More formally, we define a weighting function

$$W_{\mathrm{p}}(ij,\pi) = \min\left(\frac{S(ij,\pi)}{S_{\max}}, 1\right) * \lambda_{\mathrm{p}} \tag{3.11}$$

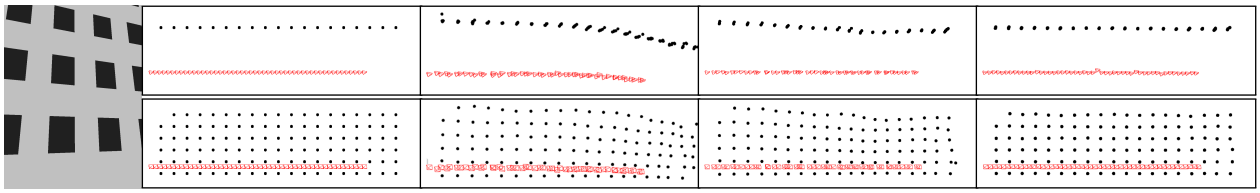
where $W_p(ij,\pi)$ defines the weight for plane π in pairwise estimate $i \rightarrow j$, $S(ij,\pi)$ returns the number of inliers in the local plane estimate, $S_{\max}$ defines the minimum number of inliers to receive full weight, and λ_{p} is the plane weighting coefficient (the maximum weight a plane can have). We use $S_{\max} = 10, \lambda_{\mathrm{p}} = 50$ in all our experiments. In Sec. 3.3.5, we show how adding plane constraints dramatically improves reconstructions of both synthetic and real-world scenes.

3.3.5 Evaluation

In this section, we present our experimental results on both a synthetic scene, where we know the ground truth results and can hence quantitatively measure accuracy, as well as some real-world hand-held videos sequences.

Synthetic scene In order to test our algorithm's (and its variants') performance, we constructed a simple synthetic 3D scene consisting of a three-story building with regularly spaced windows (Fig. 3.8a). We render this scene from a camera path translating along the length of the building, with small regular fluctuations in elevation. We bypass traditional feature extraction, and instead establish feature tracks by projecting the 3D window corners into each view. We then add synthetic 2D Gaussian noise to these tracks before passing them to the reconstruction algorithm, in order to simulate the correspondence errors which cause drift. Vanishing points are extracted normally, by running our vanishing point estimation on the rendered images. Fig. 3.8c shows that adding Gaussian noise to the 2D point tracks results in accumulated pose error, consisting of both rotational and positional (scale) drift. Fig. 3.8d shows that introducing vanishing point constraints significantly reduces the visible rotational drift (seen as bending in the reconstruction). The remaining errors, caused either

Figure 3.7: Datasets. Our real world datasets used for evaluation consist of five handheld video sequences of scenes with man-made structures. Here we show sample video frames and the approximate building facade traced on satellite imagery.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 3.8: Synthetic test case. **(a)** A sample frame from the sequence. The sequence contains three rows of "windows" on a single plane. **(b)** Top-down and frontal views of the noise-free reconstruction produced by Theia without added structural constraints: red frusta are shown for the reconstructed camera positions (moving left to right), black points denote the reconstructed 3D window corners. **(c)** After introducing noise to the 2D point tracks, the reconstruction produced by Theia exhibits both rotation and scale drift. **(d)** Once vanishing point constraints constraints are added, the rotational drift (bending) is reduced, but scale drift is still present, seen as irregularities in the frame-to-frame camera translations, and non-planarity in the reconstructed points. **(e)** Once planar constraints are added to the global position estimation, the scale drift is eliminated.

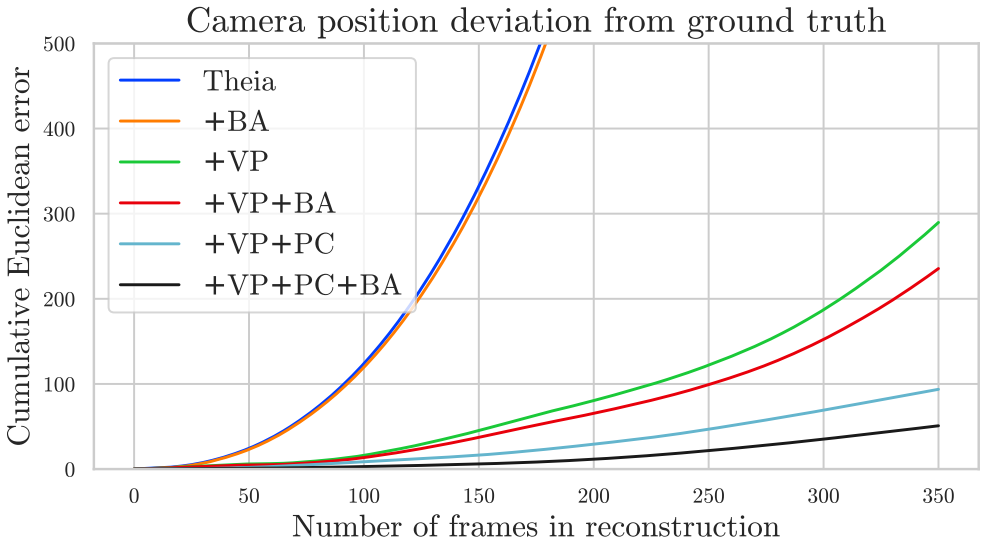


Figure 3.9: Quantitative evaluation. A comparison of positional drift across variants of Theia with global bundle adjustment (BA), our vanishing point constraints (VP), and our planar constraints (PC). One unit along the vertical axis is equal to the baseline between a pair of cameras. We see that bundle adjustment decreases error across all reconstructions, but cannot fully correct for the positional drift even after the rotational drift has been removed.

by incorrect translation directions or incorrect scale estimates, are shown to be virtually eliminated by introducing global plane constraints (Fig. 3.8e).

In Fig. 3.9, we show quantitative results, comparing the positional drift in reconstructions with and without our proposed structural constraints. We measure drift by reconstructing the synthetic sequence shown in Fig. 3.8 and comparing the reconstructed poses against ground-truth. For this experiment, we use a sample sequence with 350 frames. Since the reconstructions have scale, rotation, and translation (gauge) ambiguities, we align the reconstructed poses to the ground truth by solving a similarity transformation.

Real-world videos Typical SLAM and SfM benchmarks use static (or global shutter) cameras with wide fields-of-view and contain widely varying viewpoints of the scene. These configurations are chosen because they are the least susceptible to drift — tracks will be longer, feature positions more precise, and triangulation angles wider, thus reducing the effects of low-frequency pose error. In this section, we focus instead on the converse: configurations which are *most* susceptible, i.e. low-field of view, rolling-shutter, handheld videos, where objects are only seen from a small range of viewpoints. These types of captures also more accurately reflect the type of sequence that might be captured by a layperson with a handheld camera-phone.

With this in mind, we captured five handheld video sequences of man-made environments (Fig. 3.19). We used smartphone cameras in portrait orientation, and mostly captured sequences walking along building facades. These sequences are intended to showcase difficult configurations which typically result in drift, since point tracks do not persist for many frames, and therefore cannot establish long-term constraints. In Fig. 3.10, we show a comparison to our baseline method, as well as to COLMAP [141], a popular incremental SfM system. We can see that the baseline methods (Theia and COLMAP) both exhibit significant drift, and the addition of our constraints to Theia results in a system that produces drift-free reconstructions very quickly (a fraction of the time needed for COLMAP). It is important to note that none of these results use any form of explicit loop closure. This is intended to more visibly demonstrate the effects of drift, as automatic loop closure is not always an

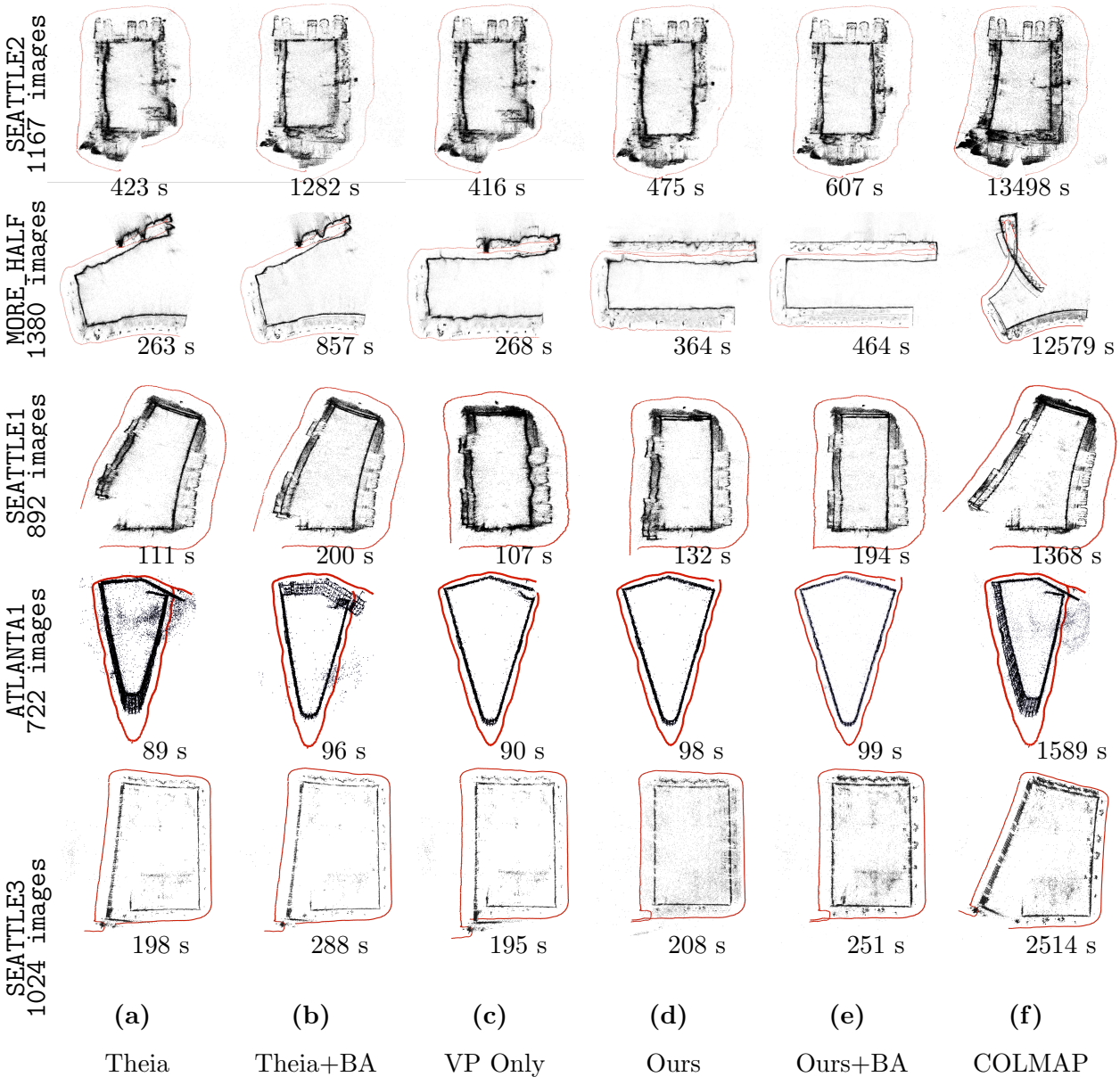


Figure 3.10: Qualitative comparison. Top-down views of the real-world datasets using different reconstruction methods: **(a)** Theia without added structural constraints or global bundle adjustment (BA), **(b)** Theia without added constraints but with BA, **(c)** Theia with added vanishing point constraints (VP), but no planar constraints (PC) or BA **(d)** Theia with both VP and PC, but without BA **(e)** Theia with VP, PC, and BA, **(f)** COLMAP, with BA. None of the reconstructions use any form of explicit loop closure; all examples use window-based feature matching with a window size of 100 frames. See supplemental for a discussion about loop closure. Inset values indicate reconstruction time, excluding feature matching, measured on a MacBook Pro with a 6-core 2.9 GHz Intel processor and 32 GB of memory.

SEATTLE1		SEATTLE2		SEATTLE3		ATLANTA1		
Position	Orientation	Position	Orientation	Position	Orientation	Position	Orientation	
73.51	12.63°	45.04	8.70°	48.06	10.23°	36.22	15.71°	Theia
30.01	12.60°	11.63	8.56°	46.85	9.19°	32.02	14.26°	Theia+BA
16.98	**1.72°**	25.00	**2.01°**	40.07	2.70°	18.11	2.18°	Theia+VP
7.42	**1.72°**	3.61	**2.01°**	4.22	2.70°	6.50	2.18°	Ours
1.91	**1.72°**	**3.34**	2.49°	**3.16**	**1.03°**	**2.98**	**1.63°**	Ours+BA
64.38	17.67°	29.38	6.74°	82.80	24.10°	31.84	15.12°	COLMAP

Table 3.1: Loop closure error. (lower is better): for sequences which end at approximately the same location, we can compute the *loop closure error* by duplicating the first image at the end of the sequence, and measuring the error between the two reconstructed views in both position and orientation. This effectively measures the amount of drift over the entire sequence. Position errors are divided by the median baseline to show the deviation in number of frames. The shown configurations are defined in Fig. 3.10.

option (for sequences without enough loop overlap) and, when it is, does not always remove the effects of drift. It is also possible for automatic loop closure to introduce other sources of error, such as false matches between distant views, especially in sequences like ours, which contain significant repetitive structure. In the supplemental material, we show experiments with automatic loop closure enabled.

In order to quantify the amount of drift for the complete-building sequences, we copied the first frame in our sequence as the last frame, but ran both Theia and COLMAP without loop closure, and measured the error between the two reconstructed frames. The quantitative errors in both orientation and position are shown in Table 3.1. While adding vanishing point constraints dramatically reduces orientation errors, the positional error is even further reduced by adding the plane constraints. Performing a final bundle adjustment on this solution even further reduces the error.

Note that our added constraints do not modify the correspondences, and thus the constraints in bundle adjustment are unchanged. However, our constraints improve the initializa-

tion to bundle adjustment, providing reconstructions often much closer to the true solution. This causes bundle adjustment to converge more quickly, and reduces the likelihood of converging to local minima. Since planar constraints are not enforced in bundle adjustment, bundle adjustment could theoretically reintroduce drift, but we have not observed this in any of our experiments.

Loop closed results Loop closure, in the context of an SfM system like Theia or COLMAP, only has significance when the matching strategy relies on temporal proximity, i.e. frames are only matched to nearby frames in the video sequence. In these cases, feature matching may not be performed between the first and last frames in a sequence, even if they observe the same parts of the scene. Loop closure addresses this by adding pairwise pose estimates between frames which may not have otherwise been matched, thus adding constraints to the rotation and position estimation stages. These pairs of frames can be identified by a number of strategies, including vocabulary trees and spatial proximity in incremental reconstruction.

When dealing with buildings that can be circumnavigated, one might think that automatic loop closure would resolve all drift error. As can be seen in Fig. 3.11, this is not the case for our sequences. For two of our sequences (`SEATTLE3`, `ATLANTA1`) the addition of automatic loop closure caused the reconstruction to collapse, due to spurious loop-closure matches of repetitive structures. For the remainder of the sequences, while introducing loop closure does indeed pull the camera centers of the first and last frames closer to one another, visible errors emerge elsewhere.

Figure 3.11 also demonstrates that our added structural constraints perform similarly even when loop closure is enabled. In fact, when comparing to the addition of loop closure, we see that our added constraints are both more performant and accurate.

In the previous experiments, we do not employ loop closure, in order to more visibly demonstrate the effects of drift (for the qualitative experiments) and to be able to quantify drift (for the quantitative experiment in Table 3.1). After all, loop closure is not always an option, as not all sequences return to the same viewpoint, and even when they do, loop closure techniques can sometimes fail to find a match, especially when closure overlap is minimal.

Furthermore, as we see in our experiments, while loop closure constraints encourage the trajectory endpoints to align, they do not guarantee the elimination of all errors resulting from drift.

In Figure 3.11, we can see that for the SEATTLE2 sequence, while both the addition of loop closure and bundle adjustment produce more reasonable looking reconstructions, the point clouds still contains significant errors, as seen in the bent walls of the building. In the top row, we see that Theia struggles to fully reconstruct the structure of the bottom right corner of the building without the help of our structural constraints. In the second row, we see that even though Theia+BA produces a connected loop, the bottom half of the building facade is significantly bent. As can be seen in column (e), our added constraints resolve both these errors. In the SEATTLE1 sequence, the addition of loop closure adds significant errors in the form of multiple discontinuities in camera position and noisy reconstructed structure. Even so, our method is able to recover a reconstruction similar in quality to the non-loop-closed variant. In both sequences, COLMAP's loop closed reconstruction simply moves the trajectory discontinuities seen in the non-loop-closed version to a different part of the sequence, whereas our results produce straight walls and continuous camera trajectories. The remaining two closed-loop sequences (SEATTLE3, ATLANTA1) contain significant repetitive structure resulting in large numbers of spurious loop closure matches. As a result, a reasonable reconstruction was not achieved through any of the shown configurations with automatic loop closure enabled.

3.3.6 Discussion and Conclusions

In this section, we have presented a method for efficiently constraining global reconstructions in scenes with man-made structures. We show that the detection and linking of *extended structural features* such as planes and vanishing points, which can span frames that may not have overlapping views of the scene, can be used as powerful additional constraints in structure from motion, enabling the reconstruction of sequences that are particularly susceptible to drift. We demonstrate that these constraints significantly reduce accumulated

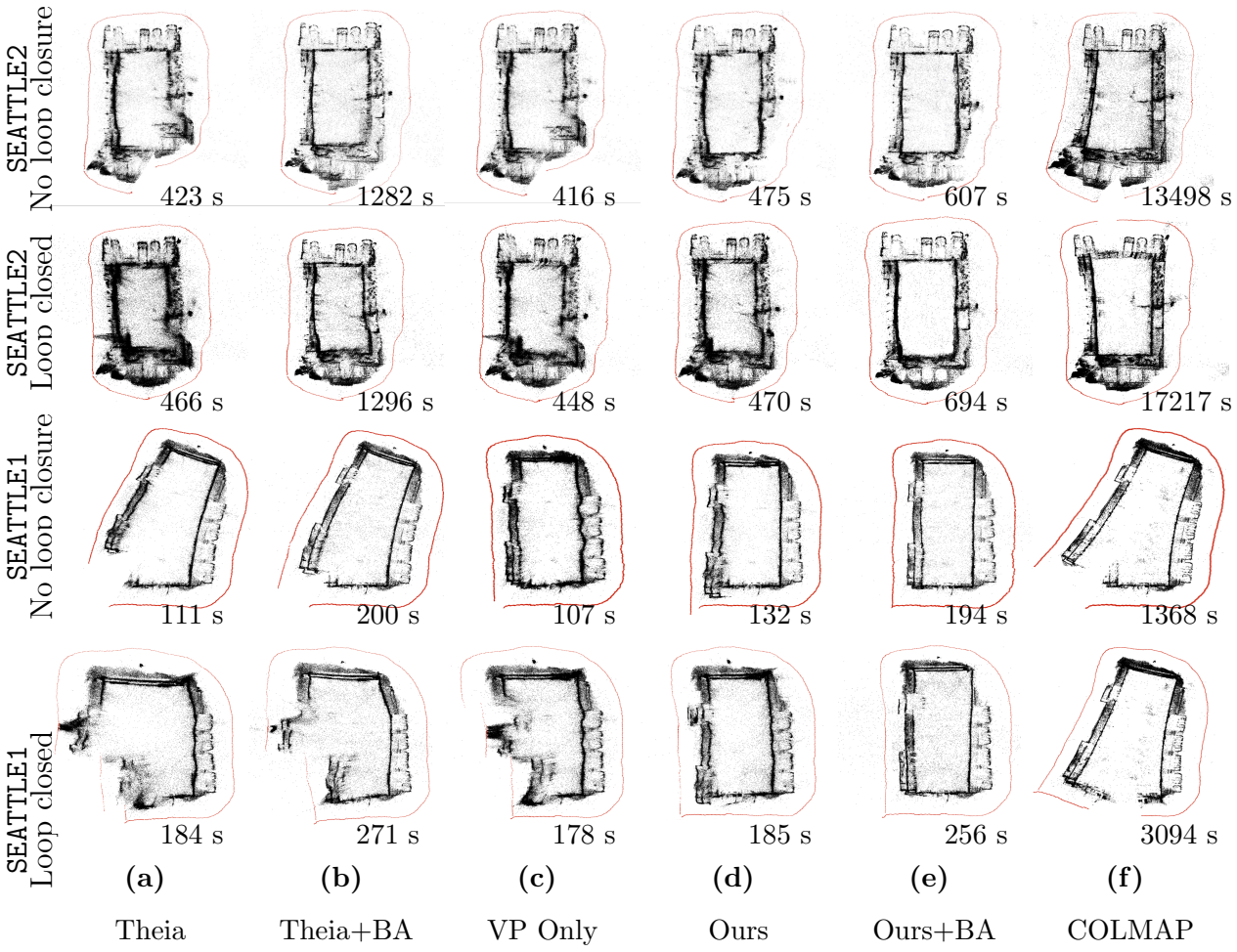
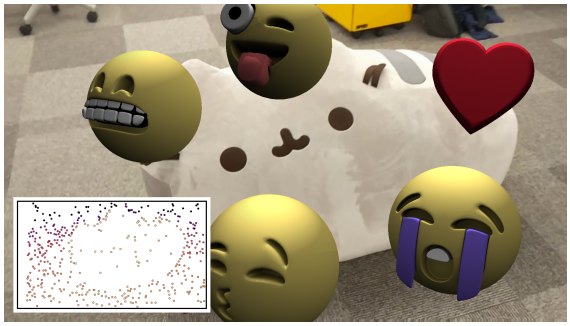
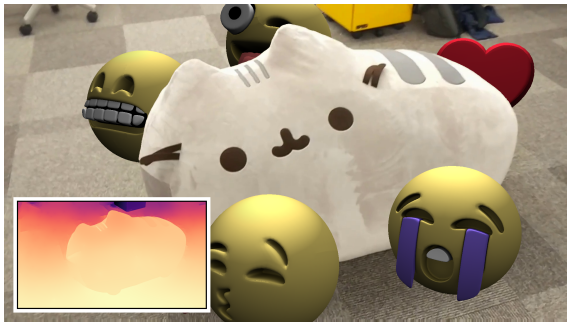


Figure 3.11: Loop closure comparison. Top-down views of the real-world datasets using different reconstruction methods: **(a)** Theia without added structural constraints or global bundle adjustment (BA), **(b)** Theia without added constraints but with BA, **(c)** Theia with added vanishing point constraints (VP), but no planar constraints (PC) or BA **(d)** Theia with both VP and PC, but without BA **(e)** Theia with VP, PC, and BA, **(f)** COLMAP, with BA. Each sequence is shown twice, both with and without automatic loop closure.



(a) AR Overlay using Sparse SLAM Reconstruction



(b) Occlusion-aware AR with Our Densification

Figure 3.12: Occlusion-aware depth estimation. Camera tracking systems estimate depth only a few sparse point features. This limits AR effects to pure overlays (a) because the scene geometry is not known for most pixels. Our technique (b) propagates the sparse depth to every pixel to produce dense depth maps. They exhibit sharp discontinuities at depth edges but are smooth everywhere else, which makes them particularly suitable for occlusion-aware AR video effects.

drift over longer sequences, allowing fast-but-brittle global reconstruction algorithms to rival — and often surpass — the accuracy of costly bundle-adjusted incremental systems in scenes with strong structural elements. We present comparisons to state-of-the-art structure-from-motion systems for both synthetic and real-world data.

3.4 Resolving misaligned depth boundaries

A popular use-case for 3D reconstruction is virtual scene interaction, often referred to as augmented reality (AR). AR is a transformative technology that can be used to enhance (even live) video streams with interactive computer generated 3D content. AR has many applications ranging from bringing virtual monsters into your bedroom (gaming), to previewing virtual furniture in your living room (shopping), or even breaking down the mechanics of your car's engine (learning), among many others.

Recent advancements in mobile hardware and tracking technology enable consumers to experience AR directly on their cell phone screens. This is typically powered by SLAM

algorithms, such as Google's ARCore[2] and Apple's ARKit[3], which track a few dozen scene points in 3D and compute the 6-DOF trajectory of the device. Once the camera pose is known, we can overlay 3D content on the image, such as face masks or artificial animated characters.

However, since the tracked points are sparse, these effects cannot interact with the scene, because the scene geometry is not known for most pixels. This means that virtual objects can never be *occluded* by real objects (e.g., the Pusheen plush doll in Figure 3.1a). The absence of occlusion is often jarring and can break the illusion of reality.

In this section we propose a method [66] that overcomes this limitation by *densifying* the sparse 3D points, so that depth is known at every pixel. This enables a much richer set of effects that fully interact with the scene geometry and make use of occlusions (Figure 3.1b).

Multi-view stereo (MVS) methods can be used to reconstruct dense geometry from multiple images, but they are often not the best tool to achieve our goal, since they often (1) are not designed for video sequences and produce temporal artifacts such as flickering, (2) produce noisy results in untextured regions, (3) leave holes when omitting unconfident pixels, (4) frequently suffer from misaligned depth edges ("edge-fattening"), and (5) are prohibitively slow.

In addition, the requirements for dense, occlusion-aware AR applications are notably different from MVS. While the primary measure that these methods optimize is geometric accuracy of depth and normals, this property is less critical for AR. Instead, we are interested in a different set of objectives:

1. Sharp discontinuities: depth edges must be sharp and well-aligned with image edges to produce convincing occlusions.
2. **Smoothness:** away from discontinuities, and in particular across texture edges, the depth should be smooth to avoid spurious intersections with virtual objects.
3. **Depth ordering:** the correct ordering of layers needs to be preserved to produce

[2]https://developers.google.com/ar/

[3]https://developer.apple.com/arkit/

correct occlusions.

4. **Temporal coherence:** the depth needs to be consistent across frames to avoid flickering.
5. **Completeness:** every pixel must have an assigned depth, so we can apply the effect everywhere.
6. **Speed:** for real-time AR effects we need to compute results at fast rates and with little delay.

We designed our method to satisfy all of these objectives. It takes the sparse points computed by a SLAM system as input and propagates their depths to the remaining pixels in a smooth, depth-edge aware, and temporally coherent fashion. The algorithm starts with computing a soft depth edge likelihood map by merging forward and backward optical flow fields using a novel reliability measure. Using a future frame causes a slight delay in the output, which depends on keyframe spacing, but is enforced to be at most 116ms in our experiments. A variant of our method can be run in a fully causal fashion, i.e., without any latency, at the expense of somewhat lower result quality. The depth edges are thinned and aligned with the image edges using an extension of the Canny edge detector that takes the soft depth edges into account. Finally, we densify the sparse points by optimizing the propagation of their depths smoothly (except at depth edges) and in a temporally coherent manner.

Our method runs at near-realtime rates on a desktop machine, and modern mobile processor benchmarks indicate that a fast mobile phone implementation should also be possible. We have tested our method on a variety of video sequences, and compared against a set of state-of-the-art baseline algorithms. We also designed evaluation metrics that capture the objectives stated above and perform extensive numerical evaluations. We demonstrate the effectiveness for AR applications with two example effects that make use of dense depth.

3.4.1 Previous Work

In this section, we highlight some of the work in related areas.

SLAM Simultaneous localization and mapping algorithms [36, 135, 37] compute the camera trajectory as well as a geometric scene representation from a video stream. This problem is related to Structure from Motion, but a fundamental difference is that SLAM techniques are optimized for realtime applications and specificially designed for video sequences. SLAM algorithms are typically used for tracking in AR applications.

Most methods, however, track only a select set of independent image features, which results in a sparse scene representation and limits AR effects to pure overlays.

Dense SLAM Some SLAM methods attempt to reconstruct a dense depth map that covers all pixels in the video [113, 105]. These methods are often slower, however, and may be less accurate (see discussion Engel et al.'s paper [36]). Similar to MVS methods (discussed below), they are also not explicitly designed to have sharp depth discontinuities that are well-aligned with image edges, which might result in artifacts at occlusion contours.

There are also intermediate, semi-dense approaches that reconstruct a subset of pixels [37]. However, these methods have the same limitation w.r.t. virtual object occlusions as sparse methods.

MVS Multi-view stereo methods [43, 144] compute dense geometry from overlapping images. However, as stated in the introduction, their depth maps are highly optimized for geometric accuracy, but might not work well for AR applications. For example, most algorithms drop uncertain pixels (e.g., in untextured areas) and edges in the estimated geometry are often not well aligned with image edges.

Video-based Depth Estimation Most stereo algorithms are not designed to produce temporally coherent results for video sequences, however, there are some exceptions. Zhang et al. [195] optimize multiple video frames jointly with an explicit geometric coherency term. However, similar to many MVS algorithms the runtime is prohibitively slow for our applications (several minutes per frame). Hosni et al. [68] use a weighted 3D box filter to spatio-temporally smooth cost volumes before extracting disparity values. Richardt et al. [136] use instead a bilateral grid to spatio-temporally smooth a cost volume. Stühmer et al [160] esti-

mate depth from multiple optical flow fields. However, optical flow estimation is unreliable in untextured regions, and it is slow at high quality settings.

Edge-aware Filtering Sparse annotations, such as depth from SLAM points, can be densified using edge-aware filtering techniques. These techniques are typically very fast and differ in the kind of smoothness constraints they impose on the solution. Levin et al. [86] propose a quadratic optimization to propagate color scribbles on a grayscale image. A similar technique can be used to propagate depth from sparse points [147]. A joint bilateral filter [131] can also be used to propagate sparse constraints while respecting intensity edges. The bilateral solver [11] can be used in a similar way, e.g., in the Google Jump system [3] it smoothes optical flow fields in an edge-aware manner. Bonneel et al. [16] extend edge-aware filtering to video and add temporal constraints to reduce flickering. Weerasekera et al. [182] use the output of a single-view depth estimation network to propagate sparse depth values. Similarly, [200] constrain the propagation of sparse depth values using the output of a neural network which predicts surface normals and occlusion boundaries. Park et al. [125] describe techniques for upsampling and hole-filling depth maps produced by active sensors using constrained optimization. A survey of additional densification methods can be found in [124]. The drawback of many of these image-guided filtering techniques is that the propagation stops not only at depth edges, but also at texture edges. This can lead to false discontinuities in the depth maps. Additionally, many of these methods are prohibitively slow for real-time AR applications. We compare our method against three of the above filtering techniques in Section 3.4.4.

3.4.2 Overview

The inputs to our algorithm are (1) a sequence of video frames, typically captured with a cell phone camera, (2) camera parameters at every frame, and (3) sparse depth annotation (at least for some frames). We use an existing SLAM system [36] to compute (2) and (3). Our algorithm propagates the sparse depth annotation to every pixel in near realtime and

with only a short delay of a few frames.

As mentioned before, the criteria that make depth maps desirable for AR applications are different from the goals that most MVS algorithms are optimized for, since we neither require correct *absolute* (or metric) depth, nor do we require perfect object normals. Rather, our algorithm is designed to produce *approximately* correct depth that satisfies the criteria stated at the beginning of this section:

1. **Discontinuities:** we estimate the location of depth edges and produce sharp discontinuities across them.
2. **Smoothness:** our depth maps are smooth everywhere else, in particular across texture edges.
3. **Temporal coherence:** we optimize consistency of depth over time.
4. **Completeness:** by design we propagate depth to every pixel.
5. **Speed:** our algorithm takes on average 48.4ms per frame and the output is delayed by at most 116ms in the complete method, and has no delay in the causal variant.

Our algorithm proceeds in three stages:

1. *Estimate soft depth edges (Figure 3.13d):* First, we examine the gradient of optical flow fields to estimate "soft" depth edges that are not well-localized, yet. Because optical flow is unreliable near occlusions we compute flow fields to a future and past frame (Figure 3.13c) and fuse the resulting depth edges using the observation that edges in the flow gradient are only reliable when the flow vectors are diverging.

2. *Localize depth edges (Figure 3.13e):* Next, we localize the depth edges using a modified version of the Canny edge detector [22]. This procedure thins the edges and aligns them with the image gradient, so that they are precisely localized on the center of image edges. It uses hysteresis to avoid fluctuations in weak response regions.

3. *Densification (Figure 3.13f):* Finally, we propagate the sparse input depth to every

pixel by solving a Poisson problem. The data term of the problem is designed to approximate the sparse input depth and to encourage temporal continuity, while the smoothness term encourages sharp discontinuities at the detected depth edges and a smooth result everywhere else.

3.4.3 Method

Camera Parameters and Sparse Depth The first stage of our algorithm computes a "sparse reconstruction" using a SLAM system. This computes two entities that are required for the subsequent stages:

1. Extrinsic camera parameters for every frame (i.e., rotation and translation); we assume the intrinsic parameters are known.

2. Sparse 3D scene points (Figure 3.13b). Our algorithm will propagate their depth to the remaining pixels in a later stage to generate the dense result.

We experimented with various existing systems and settled on using DSO-SLAM [36], since it is fast and robust. Since DSO only provides 3D points at intermittent key frames, we reproject these to the surrounding non-key frames, so every frame has a sparse depth source.

Soft Depth Edges The goal of this stage is to find "soft" depth edges, which means that they are defined by a continuous strength value and do not need to be accurately localized (Figure 3.13d). They will be thinned, binarized, and aligned with the image edges in the next section. For performance reasons we compute the soft edges on downscaled images (1/4 in each dimension) and upscale the results at the end of the stage.

We start by selecting a nearby frame with sufficiently large baseline to the current frame, and compute a dense optical flow field. In the flow field we can identify depth edges at places where the gradient magnitude is high, because sudden changes in depth imply corresponding changes in the flow, due to parallax (Figure 3.14, bottom row).

Unfortunately, optical flow is unreliable around pixels that are occluded in the nearby frame (compare Figures 3.14d and 3.14f with Figure 3.14e and note how only one of the two edges of the object are resolved in each flow image, respectively). We alleviate the situation by computing flow w.r.t. *two* nearby frames, one backward and one forward in time. These tend to have different sets of pixels that are unreliable (see manual annotations in Figure 3.14e). We fuse the two depth edge maps using a novel optical flow reliability measure, described below, to achieve a result that contains the correct edges from both (Figure 3.14h). A comparison of the reliability merging and the more naive approach of taking the per-element gradient maximum can be found in the supplementary material. We also provide comparisons with off-the-shelf boundary estimation methods, such as [33].

Selecting Nearby Frames The nearby frames need to provide sufficient translational motion so we achieve a strong flow gradient response and reduce noise. There are many sensible ways in which these could be selected; we use the following simple heuristic. We compute the spatial distance between the camera positions of the two DSO key frames that bracket the current frame. Then, we look forward and backward from the current frame and select the first frame in either direction whose camera position is at least half that distance away.

Optical Flow We compute optical flow using DIS Flow [82], because it is one of the fastest available methods. We use the implementation in OpenCV and chose the `ultrafast` preset. Since the results exhibit a block pattern we smooth it using a 7×7 median filter.

Computing the Gradient Magnitude Let F be one of the two flow images from the current frame to one of the nearby frames. We compute the gradient magnitude M, at every pixel p selecting the x or y component, whichever provides the higher value:

$$M(p) = \max\Big(\big\|\nabla F_x(p)\big\|_1,\ \big\|\nabla F_y(p)\big\|_1\Big). \tag{3.12}$$

Fusing Forward and Backward Results As described above, different sets of pixels are reliable in the two flow fields. We observe that places where the edge-normal flow projections

f are *diverging* are generally more reliable; this situation occurs at *disocclusions*, when both pixels are visible in the nearby image. The opposite is true for *converging* flow projections; this indicates an *occlusion*: one of the two pixels is not visible in the nearby image and therefore its flow vector cannot be accurately estimated.

We turn this observation into a per-pixel reliability score as follows. Given a pixel of interest p, we find two helper pixels p_0 and p_1 that are offset at unit distance in the gradient direction d and its opposite. We compute the projection of the flow vectors at p_0 and p_1 on d:

$$f_0 = F(p_0) \cdot d, \quad f_1 = F(p_1) \cdot d, \tag{3.13}$$

and obtain the reliability score as their difference,

$$r = f_1 - f_0. \tag{3.14}$$

r will be positive (reliable) for diverging flow projections, and negative (unreliable) for converging flow projections. Figure 3.15 illustrates this on two examples.

Now we can fuse the gradient magnitude from both nearby images by selecting at each pixel the more reliable quantity:

$$M_F(p) = \begin{cases} M_{prev}(p), \text{if } r_{prev}(p) > r_{next}(p) \\ M_{next}(p), \text{else} \end{cases} \tag{3.15}$$

Filtering The fused gradient magnitude M_F identifies depth edges well without getting confused by texture edges (Figure 3.16a-b). However, they are not well aligned with color images. To make the alignment with image edges in the following stage easier, we blur M_F with a wide box filter of size $k_F = 31$, to ensure edges overlap with their corresponding image edges.

We also suppress noise by applying a temporal median filter that includes samples from $k_T = 7$ frames. We determine the temporal neighbors of pixels by estimating a per-frame homography warp from the SLAM points (using the OpenCV `findHomography()` function).

Finally, we normalize M_F by dividing it by the 90^{th} percentile value. This makes the parameter settings more invariant to the video content.

The final spatio-temporally filtered soft depth edges $\widetilde{M}_F$ are shown in Figure 3.16c.

Localizing the Depth Edges In this section, we accurately localize the depth edges by thinning and binarizing them, and aligning them with the color image edges. We achieve these goals by modifying the Canny edge detector [22], which performs a similar operation just on intensity image edges. Recall the basic operation of the Canny detector:

(1) **Intensity gradient magnitude:** Compute the (blurred) gradient magnitude of the intensity image $\widetilde{M}_I$ (we normalize it by dividing by the 90^{th} percentile intensity).

(2) **Non-maximum suppression:** all values of $\widetilde{M}_I$ except the local maxima are suppressed, to thin edges down to a width of a single pixel.

(3) **Double thresholding:** Using two thresholds τ_{high} and τ_{low} the edge pixels are classified into strong ($\widetilde{M}_I > \tau_{high}$), weak ($\tau_{high} \geq \widetilde{M}_I \geq \tau_{low}$), and suppressed edge pixels ($\tau_{low} > \widetilde{M}_I$).

(4) **Hysteresis:** Every strong edge pixel is selected, but weak edge pixels are only selected when they are connected to a strong edge. This effectively removes spurious weak edge pixels.

Figure 3.17c shows the result of applying the Canny detector on a color image. The detector identifies and localizes all intensity edges well. Notably, this set of edges includes also all major depth edges. This indicates that we can achieve our goal by selectively suppressing only the texture edges while keeping the depth edges.

We do so by injecting another threshold on the soft depth edge map $\widetilde{M}_F$ into the algorithm. More precisely, we change the definition of a strong edge pixel to require not just a high intensity gradient response ($\widetilde{M}_I > \tau_{high}$) but also a high response in the soft edge map ($\widetilde{M}_F > \tau_{flow}$). The definition of weak and suppressed edge pixels remain unchanged.

With this modification we start depth edges only where the soft edge map indicates a high confidence, but we allow continuing them into low soft edge response regions as long as there remains a sufficiently strong image edge. This helps bridging over gaps in the soft edge map due to noise in the optical flow image. Figure 3.17e shows the result of the modified detector. It effectively preserves most of the depth edges while suppressing most texture edges.

Densification In the final stage we use the localized depth edges to control the propagation of sparse SLAM point depths to the remaining pixels. We set this up as a quadratic optimization problem, using the following constraint terms.

A unary data term encourages approximating the depth of the SLAM points:

$$E_{data}(p) \ = w_{sparse}(p) \ \big\|D(p) - D_{sparse}(p)\big\|_2^2. \tag{3.16}$$

D_{sparse} is a depth map obtained by splatting the depths of SLAM points into single pixels, and w_{sparse} is 1 for all pixels that overlap a SLAM point and 0 everywhere else.

A second unary data term encourages the solution to be temporally coherent:

$$E_{temp}(p) \ = w_{temp}(p) \ \big\|D(p) - D_{temp}(p)\big\|_2^2. \tag{3.17}$$

D_{temp} is a depth map obtained by reprojecting the (dense) pixels of the previous frame using the projection matrices estimated by the SLAM system. w_{temp} is 1 for all pixels that overlap a reprojected point and 0 everywhere else (splatting gaps, near boundaries).

We use a spatially varying pairwise smoothness term:

$$E_{smooth}(p,q) = w_{pq} \left\|D(p) - D(q)\right\|_2^2, \tag{3.18}$$

with the weight

$$w_{pq} = \begin{cases} 0, & \text{if} \quad B(p) + B(q) = 1, \\ \max\big(1 - \min(s_p, s_q), 0\big), & \text{else.} \end{cases} \tag{3.19}$$

B denotes the binarized depth edge map, computed in the previous section, and $s_p = \big(\widetilde{M_F} \cdot \widetilde{M_I}\big)(p)$, $s_q = \big(\widetilde{M_F} \cdot \widetilde{M_I}\big)(q)$. At depth edges we set the weight to zero to allow the values to drift apart without any penalty and form a crisp discontinuity. Everywhere else, we enforce high smoothness if either $\widetilde{M_I}$ is low (textureless regions) or $\widetilde{M_F}$ is low (possibly texture edge but not a depth edge).

We obtain the following combined continuous quadratic optimization problem:

$$\arg\min D \quad \lambda_d \sum_p E_{data}(p) + \lambda_t \sum_p E_{temp}(p) + \lambda_s \sum_{(p,q)\in N} E_{smooth}(p,q), \tag{3.20}$$

where N is the set of horizontally and vertically neighboring pixels. We use the balancing coefficients $\lambda_d = 1, \lambda_t = 0.01, \lambda_s = 1$.

The solution to Equation 3.20 is a set of sparse linear equations. It is, in fact, a standard Poisson problem, for which we have specialized solvers that can optimize it rapidly. We use an implementation of the LAHBF solver [165] to optimize it.

Figure 3.18 shows the impact of suppressing texture edges in the densification. When using standard Canny intensity edges, the resulting depth maps contains many false depth discontinuities (Figure 3.18c). These are mostly absent in our result (Figure 3.18e).

3.4.4 Results & Evaluation

While our method is ultimately intended to be run in a real-time setting, e.g., in the viewfinder of a smart phone, we implemented it in practice to operate on pre-captured video sequences, since it enables easier debugging and more reproducible results.

We captured and processed a number of video sequences with a Google Pixel 2 smart phone at full HD resolution (1920×1080 pixels). Screenshots from each sequence can be seen in Figure 3.19. The videos contain indoor and outdoor locations, of a variety of objects and scenes, often including objects that are hard to reconstruct with traditional multi-view stereo algorithms, such as moving objects, water, reflective surfaces, and thin structures.

In the supplementary material we provide the full set of input videos, final depth maps, as well as videos of the intermediate SLAM points, soft depth edges, and localized depth edges, in form of a web page for convenient inspection.

Effects We implemented two AR effects (Figure 3.20) that make use of occlusions and the ability to interact with the dense scene geometry:

1. **Object insertion:** place virtual objects in the scene that can be occluded by real objects.
2. **Lighting effect:** insert a point light source that shades the scene with radial fall-off lighting.

Algorithm Step	Duration
Sparse Reconstruction (DSO-SLAM)	13.8ms
Optical flow, for past and future frame (DIS-Flow)	0.4ms
Soft depth edges	0.2ms
Localized depth edges	12.6ms
Densification	21.3ms
Total	**48.4ms**

Table 3.2: Timing

Breakdown of the average per-frame timings of the algorithm stages.

In the supplementary material we demonstrate each effect on several videos.

Performance All results were generated on a PC with 3.4 GHz 6-core Intel i7-6800K CPU. Our algorithm only uses the CPU. Our current implementation processes our 2-megapixel HD videos at an average of 48.3ms per frame. Table 3.2 breaks down the timings for various algorithm stages.

While our current implementation is on a desktop computer, a fast implementation on a phone seems possible for the following reasons: (1) Real-time SLAM has been demonstrated by ARKit and ARCore. (2) Well optimized Canny runs faster on modern-generation mobile phones (e.g. Google Pixel 2XL and Apple iPhone X) than our implementation on a desktop[4]. (3) Poisson systems such as Eq. 3.20 can be solved with highly specialized solvers, and real-time speeds were achieved a decade ago by [107], using an evaluation system that is less powerful than today's phones.

Since these steps comprise over 98% of our runtime, we believe an optimized phone implementation of our method can achieve real-time speeds as well.

[4]https://browser.geekbench.com/v4/cpu/9747825

Evaluation Metrics Most MVS and other depth reconstruction algorithms are optimized for geometric accuracy. However, as mentioned before, for our AR effects application we have different priorities; see the objectives stated in Section 4.1.2.

In order to quantitatively assess our method and objectively compare our method to other baseline algorithms below, we propose three evaluation metrics that capture how well these objectives are achieved:

(1) *Occlusion error* penalizes depth edges not being sharp
(2) *Texture error:* penalizes depth at texture edges not being smooth
(3) *Temporal instability:* penalizes temporal jitter of static points

These metrics correspond directly to the first three objectives stated in Section 4.1.2. The other two objectives, *completeness* and *speed* are satisfied by design: our results are always 100% complete and our method operates at near real-time rates. We describe the three metrics below, and use them in a comparative analysis in the next section.

Annotations For evaluating the occlusion and texture error we need ground truth annotations of such occurrences, which we generated as follows. We selected five datasets (Bones, Cubes, Cutting Board, Pusheen, Shoes) and from each five random images, for a total of 25 images. For each image we computed a high quality offline MVS reconstruction [142]. We then computed Canny edges and classify each edge pixel as "occlusion", "texture" or "no edge" based on its depth profile. More precisely, we compute the median depth of 5 unit-spaced pixels on either side perpendicular to the edge. If that ratio of median depths is between 1 and 1.05 we consider it a texture edge pixel, if it is above 1.2 we consider it an occlusion edge pixel, and otherwise we ignore that pixel.

We then recruited five volunteers and asked them to clean up any errors in the automatic classification. They could only erase edges but not add new ones. Each volunteer processed five images, one from each dataset. The final annotations are included in the supplementary material.

Occlusion Error This error measures for annotated occlusion pixels how crisp and well localized the edge in the depth map is. We extract a profile of 10 depth samples $\{d_i\}$ perpendicular to the edge, 5 on either side, and measure the deviation from an ideal step edge, after removing the mean and standard deviation:

$$E_{occ} = \frac{1}{N} \sum_i \left(\tfrac{d_i - \mu}{\sigma} - s_i \right)^2, \tag{3.21}$$

where μ and σ are the mean and standard deviation, respectively, and $s_i = \begin{cases} -1, \text{ if } i \leq N/2 \\ +1, \text{ else} \end{cases}$ is a step function.

Texture Error Texture edges are color changes in regions that are not occlusion boundaries. We expect the depth map to be smooth here, because false depth discontinuities would cause self-occlusion artifacts or cracks in objects.

The texture error measures for annotated texture edge pixels how much the depth profile deviates from a flat profile:

$$E_{tex} = \frac{1}{N} \sum_i \left(\tfrac{d_i - \mu}{\mu} \right)^2. \tag{3.22}$$

Note, that, unlike in Eq. 3.21 we are not dividing by the standard deviation, since this would amplify the flat profiles. Instead, we divide by the mean depth to make the error invariant to the scene scale.

Temporal Stability Error Abrupt depth changes in the video can cause flickering when rendering effects. The temporal stability error penalizes variation in the 3D position of static scene points. For the five evaluation datasets we track about 100 points on the middle 100 frames with a KLT tracker, and keep all tracks that span all frames. The error is defined as the variance of the 3D positions that are obtained when unprojecting the points using the depth map:

$$E_{ts} = \frac{1}{N} \sum_{f=1}^{100} \Big(U_f\big(p_f, D_f(p_f)\big) - \mu \Big)^2. \tag{3.23}$$

Method	E_{comb}
Bilateral Filter	37.11
Bilateral Solver	4.05
Color Constraints	3.46
Our Result	**2.54**

Table 3.3: Quantitative error. A comparison of combined error values (lower is better).

p_f is the tracked point in frame f, D_f is the depth map for the frame, and U_f is the unprojection function, which takes a 2D image coordinate and depth and returns the corresponding 3D world position. $\mu = \frac{1}{N}\sum_f U_f\big(p_f, D_f(p_f)\big)$ is the mean 3D world position.

Combined Error It is useful for parameter tuning to have a single combined scalar error that balances the various objectives. Since we consider all three metrics equally important, we determine coefficients that balances out their relative scales:

$$E_{comb} = 0.7\ \widetilde{E_{occ}}\ +\ 65\ \widetilde{E_{tex}}\ +\ 200\ \widetilde{E_{ts}}, \tag{3.24}$$

where $\widetilde{\cdot}$ indicates the median across all annotated samples. A comparison to the baseline methods can be seen in Table 3.3. We obtained these coefficients by iteratively tuning our method for each metric separately, and then taking the value that maps the median of each metric to 1.

Comparative Evaluation We compared our algorithm to various baselines using the metrics defined in the previous section.

Bilateral Solver We compare against the fast bilateral solver [11], using the publicly available implementation[5]. We use the color frames as reference image for the bilateral solver, and set the target image t and confidence image c as follows:

$$\big(t, c\big)(p) = \begin{cases} \big(D_{sparse}(p),\ w_{sparse}(p)\big), & \text{if } w_{sparse}(p) > 0, \\ \big(D_{temp}(p),\ \lambda^{bs}_{temp} w_{temp}(p)\big), & \text{else,} \end{cases} \tag{3.25}$$

[5] `https://github.com/poolio/bilateral_solver`

i.e., for pixels that fall under a SLAM point we use that point's depth as target with a confidence of one, and for all other pixels we use the reprojected points from the previous frame with a lower confidence λ^{bs}_{temp}, to make the result more temporally stable.

We tune the bilateral solver parameters as well as the temporal parameter λ^{bs}_{temp} to minimize the combined error E_{comb} and obtain the following settings:

$$\lambda = 1, \quad \sigma_{xy} = 5, \quad \sigma_i = 15, \quad \sigma_{uv} = 10, \quad \lambda^{bs}_{temp} = 0.8. \tag{3.26}$$

Bilateral Filter We compare against a joint bilateral median filter [131] guided by the color frames. Because the SLAM points are very sparse we increase the kernel size in increments of 10 pixels until there are at least 8 depth inside. To make the filter temporally coherent we also include samples from D_{temp} in a 10x10 kernel, weighted by a temporal parameter λ^{bf}_{temp}. To better preserve hard edges we use a median.

We tune the bilateral filter parameters as well as the temporal parameter λ^{bf}_{temp} to minimize the combined error E_{comb} and obtain the following settings:

$$\sigma_{spatial} = 10, \quad \sigma_{color} = 10, \quad \lambda^{bf}_{temp} = 0.85. \tag{3.27}$$

Color-based optimization constraints We also compare against a variant of our densification that uses only color-based constraints instead of our estimated depth edges. In Eq. 3.20 we replace the E_{smooth} with the pairwise term by Levin et al. [86].

We fix $\lambda_s = 1$ and tune the remaining modified densification parameters to minimize the combined error E_{comb} and obtain the following settings:

$$\lambda_d = 0.1, \quad \lambda_t = 0.1, \quad \sigma_{\mathbf{r}} = 0.01. \tag{3.28}$$

Discussion We tuned our method as well as the baselines to minimize the combined error (Eq. 3.24). Then, we evaluated the individual metrics on the five evaluation datasets, and plot all samples in Figure 3.21. Our method provides a substantial improvement over the baselines in all metrics. For a qualitative comparison, please refer to the supplementary material.

Parameter			Description
k_F	=	31	Flow gradient box filter size
k_T	=	7	Temporal median window size
k_I	=	5	Image box filter size
τ_{high}	=	0.04	Canny image high threshold
τ_{low}	=	0.01	Canny image low threshold
τ_{flow}	=	0.3	Canny flow threshold
λ_d	=	1	Balancing coefficients
λ_t	=	0.01	
λ_s	=	1	

Table 3.4: Parameters. Parameters of our algorithm. We used the default settings provided here for all results.

Parameters In Table 3.4 we lists all the parameters of our algorithm and their default settings. We omit parameters of the DSO SLAM and DIS-Flow components, since we did not change them from their default settings. All results shown in this section and the supplementary material were generated with the same settings.

We tuned our method by minimizing the combined error E_{comb}, iteratively fixing some groups of parameters while varying others until we reached a local minimum.

Limitations Our method has a number of failure cases inherited from the underlying SLAM algorithm:

1. **View-dependent appearance:** Similar to most 3D reconstruction methods our method has problems dealing with reflective and specular materials. This is actually a limitation of the SLAM component, which produces points at incorrect depths in these cases.
2. **Small translation:** The SLAM algorithm requires sufficient amount of translational motion to produce 3D points.

3. **Textureless surfaces:** The SLAM algorithm requires textured surfaces in order to accurately localize and track 3D points.
4. **Missing SLAM points:** We cannot recover the depth for objects that stick out from their surrounding but do not have any SLAM points on them. This problem is very present in the Tile Wall dataset, which misses points on the foreground stone railing.
5. **Dynamic scenes:** Our method tolerates slight scene motion, such as in the Faces dataset, but does not produce good results in the presence for highly dynamic or transient objects. Nevertheless, our method is quick to recover once the dynamic objects have stabilized. An example of this can be seen in the Walkup dataset.

Additionally, our algorithm has its own limitations that lead to interesting avenues for future work:

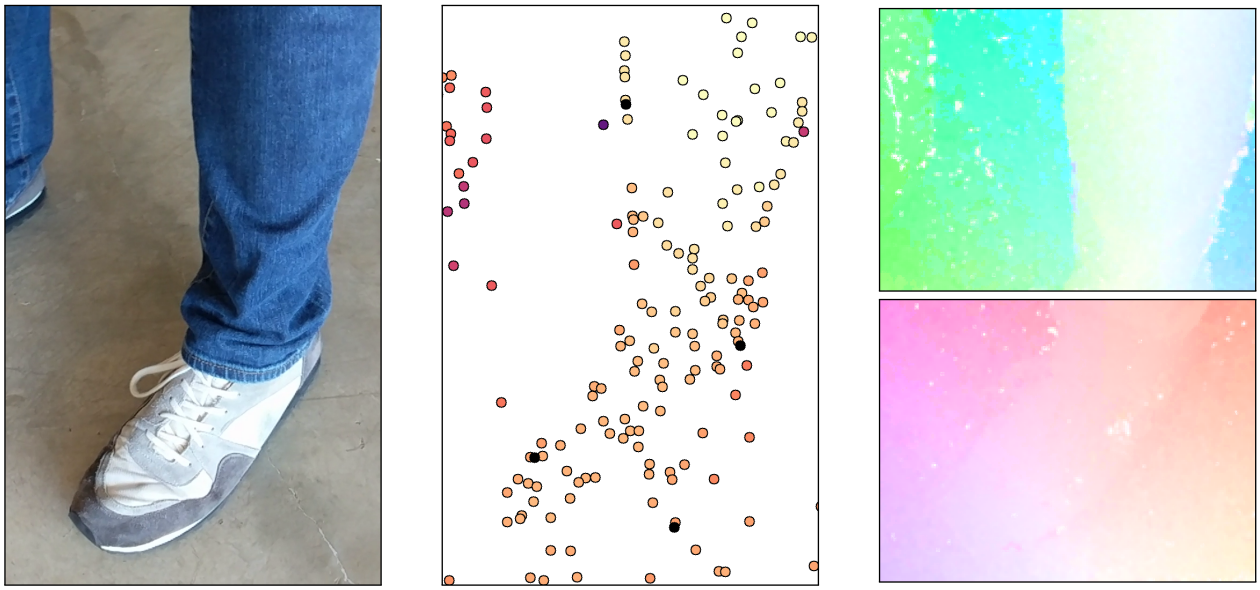
1. **Delay:** Because we are computing optical flow to a future frame our method is not fully causal but outputs results with a slight delay (depending on the spacing of key frames). In practice, we find that under regular camera motion, optimal quality can be achieved while still enforcing the lookahead to never exceed 7 frames (or 116ms at 60Hz). We have also experimented with a causal variant of our method, which uses only previous frames. Examples can be seen in the Alley, Georgian, and Lamppost datasets, which all include a portion of the video where the camera stops moving, i.e. points at which there is no future keyframe to use as a flow reference. In these cases, the resulting depth video does not suffer greatly in quality, as a result of the temporal constraints, filtering, and initialization.
2. **Low geometric accuracy:** Our method is not suitable for applications that require high geometric accuracy. It is, for example, not suitable for lighting effects that rely on accurate scene normals.
3. **Floating layers:** In cases where an object has an occlusion edge on only one side, but has strong texture edges on all sides, the Canny algorithm may trace around the entire object. This usually does not affect the final depth map, except in cases where

there are very few SLAM points on the occluding object, resulting in a "floating layer" effect. An example can be found in the SHOES sequence, where the bottom of the feet seem to be floating in front of the floor.

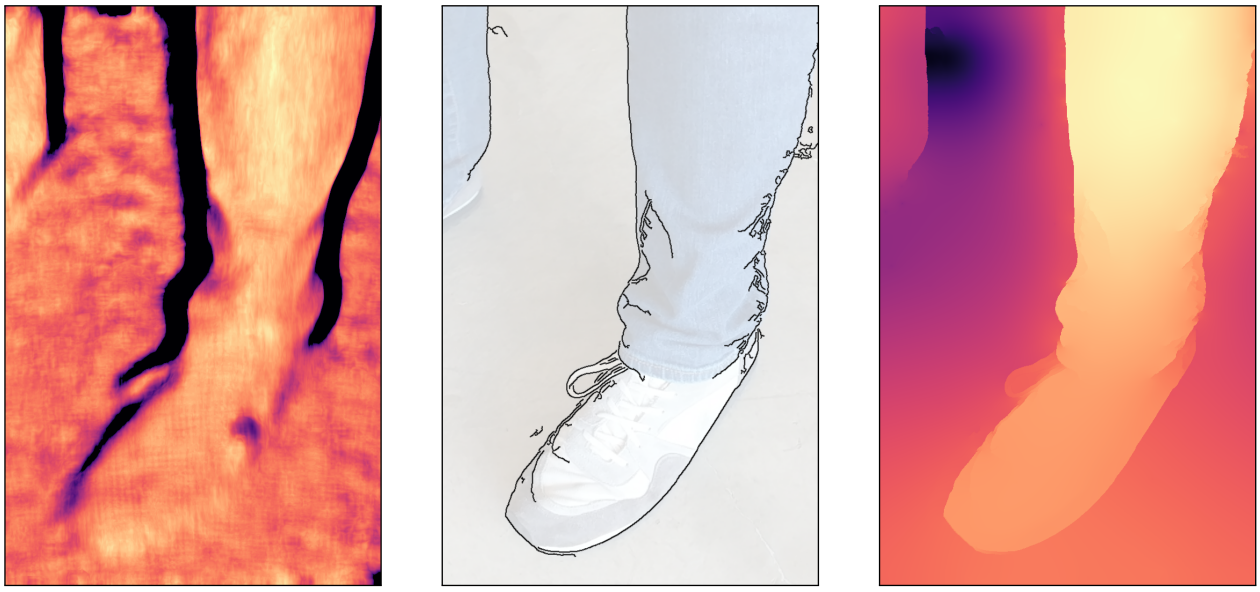
3.4.5 Conclusion

In this section we have presented a fast algorithm for propagating a sparse depth source, such as SLAM points, to all remaining pixels in a video sequence. The resulting dense depth video is spatio-temporally smooth except at depth edges where it exhibits sharp discontinuities.

These properties make our algorithm particularly useful for AR video effects. Due to the absence of holes in the depth maps, the effects can fully interact with the scene geometry, and, for example, be occluded by real objects.

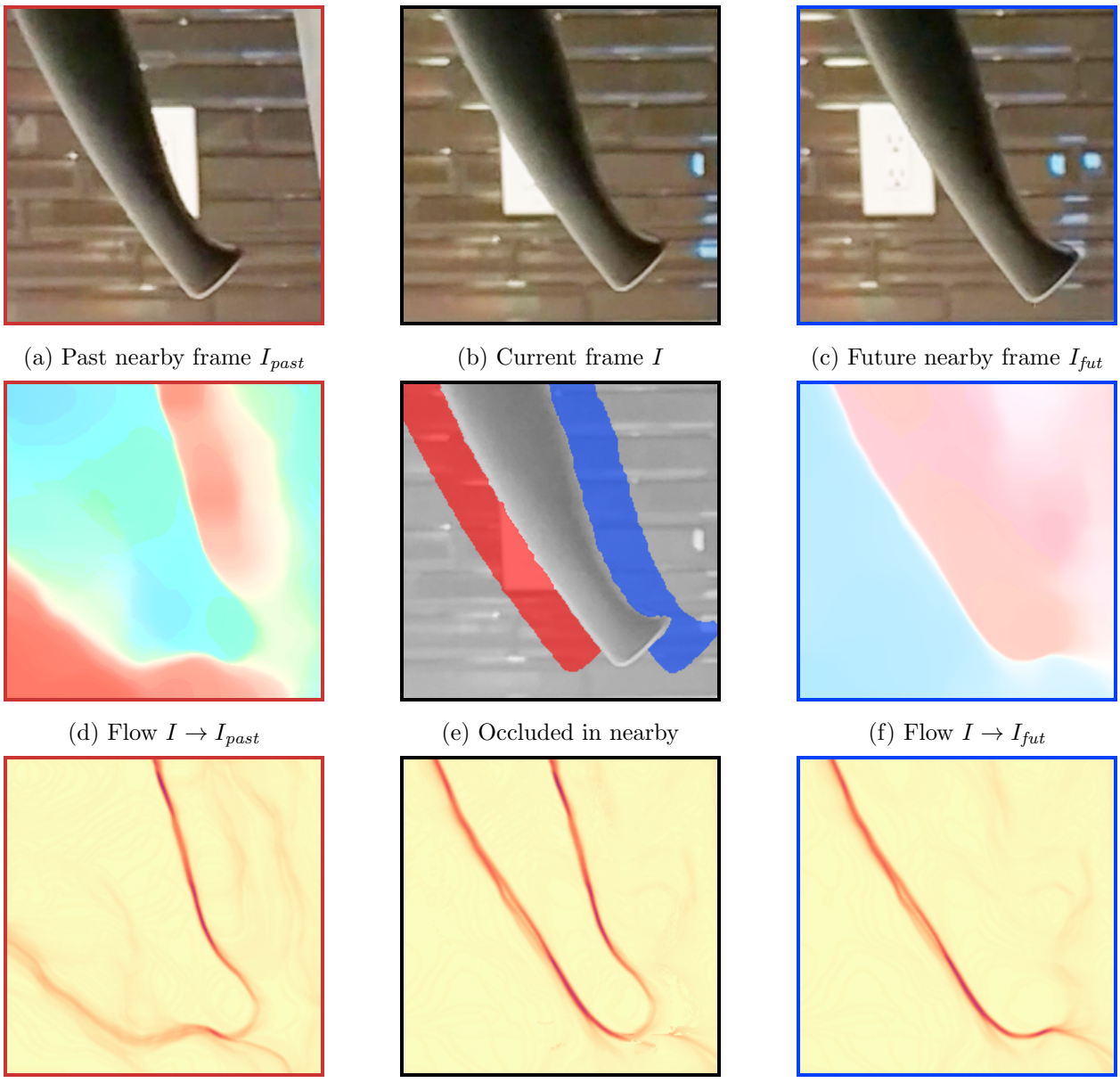


(a) Input frame (FHD, 2MP) **(b)** SLAM points (13.8ms) **(c)** Flow estimation (0.4ms)



(d) Soft depth edges (0.2ms) **(e)** Occlusion edges (12.6ms) **(f)** Final depth map (21.3ms)

Figure 3.13: Overview. Overview of the major algorithm stages. Given an input video (a) we use existing methods to compute sparse SLAM points (b) and optical flow to a nearby future and past frame (c). We fuse the most reliable responses from the forward and backward flow fields and compute "soft" depth edges that are not well-localized, and then localize and binarize these estimates to produce hard occlusion boundaries (e). Finally, we propagate the sparse SLAM point depths to every pixel in an edge-aware manner (f).



(a) Past nearby frame I_{past} (b) Current frame I (c) Future nearby frame I_{fut}

(d) Flow $I \rightarrow I_{past}$ (e) Occluded in nearby (f) Flow $I \rightarrow I_{fut}$

(g) Past grad-mag M_{past} (h) Reliability-fused M_F (i) Future grad-mag M_{fut}

Figure 3.14: Identifying and merging reliable boundary cues. For a current frame from the KITCHEN sequence (b) we select two surrounding nearby frames (a,c) and compute optical flow (d,f). The flow fields are unreliable near pixels in the current frame that are occluded in the nearby frames (e, with manual annotation for illustration). The depth edges from the corresponding gradient magnitude images are unreliable as well (g,i). Using our reliability measure (see Figure 3.15) we can compute a fused result that contains only the most reliable pixels (h).

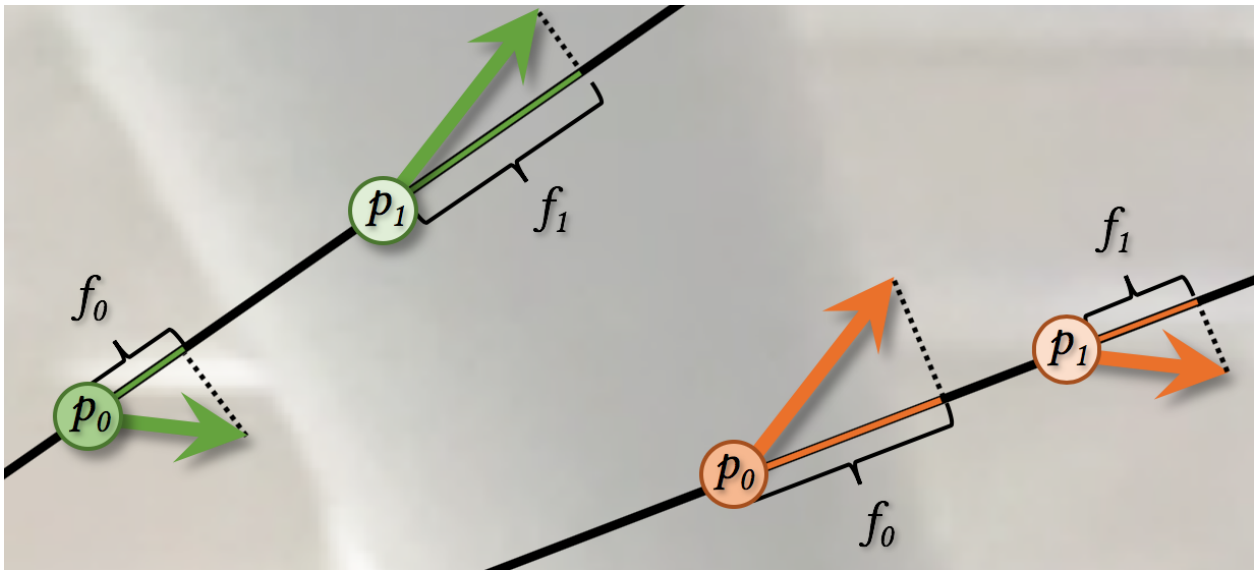
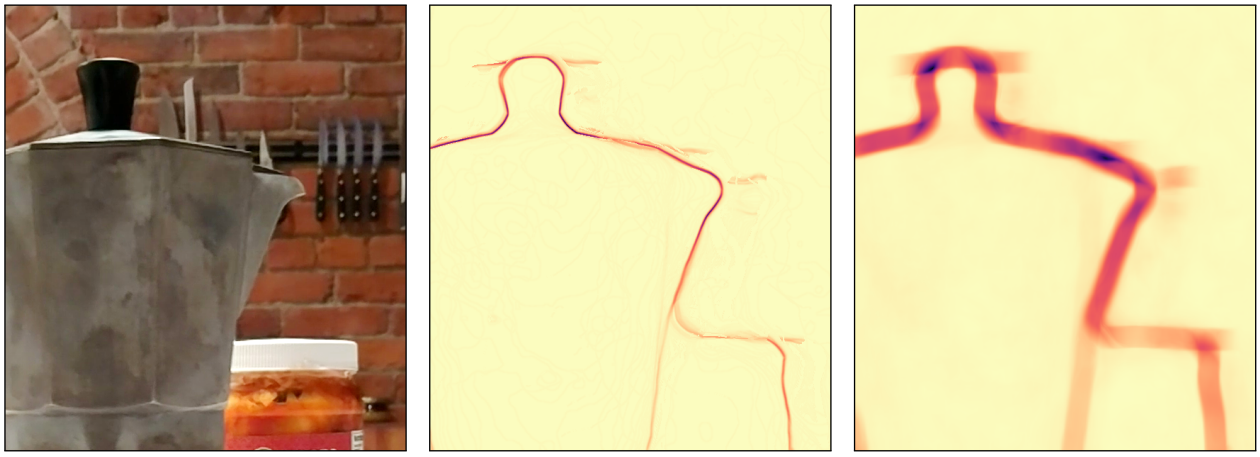
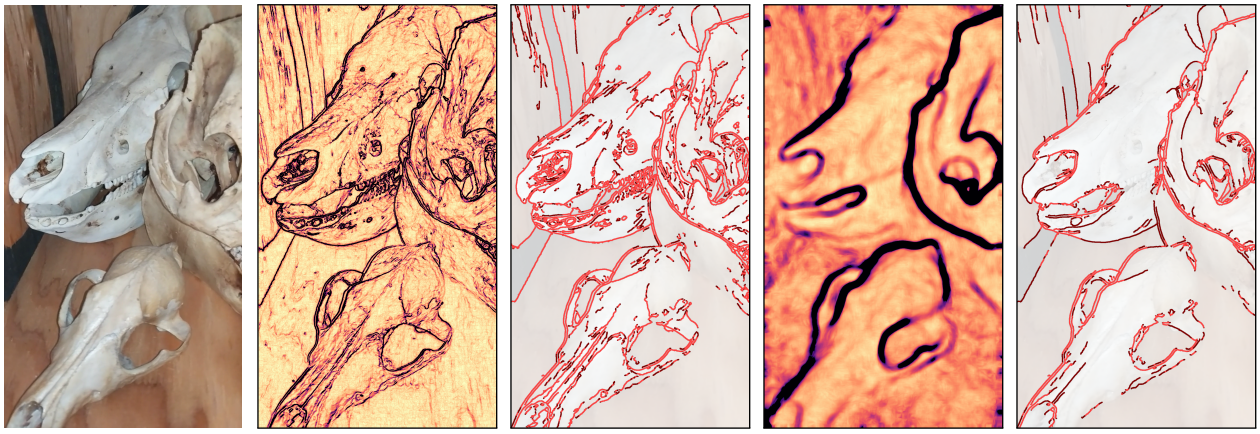


Figure 3.15: Flow gradient reliability. Our flow gradient reliability measure explained on two examples on a crop from Figure 3.14. We measure the flow at two helper pixels p_0, p_1 on either side of the edge, and compute the projections f_0, f_1 onto the line perpendicular to the gradient. Left example: the flow projections are diverging ($f_1 - f_0 > 0$), which indicates a reliable edge, since both pixels are visible in the nearby image. Right example: the flow projections are converging ($f_1 - f_0 < 0$), which indicates an unreliable edge, since at least one of the pixels might not be visible in the nearby image.



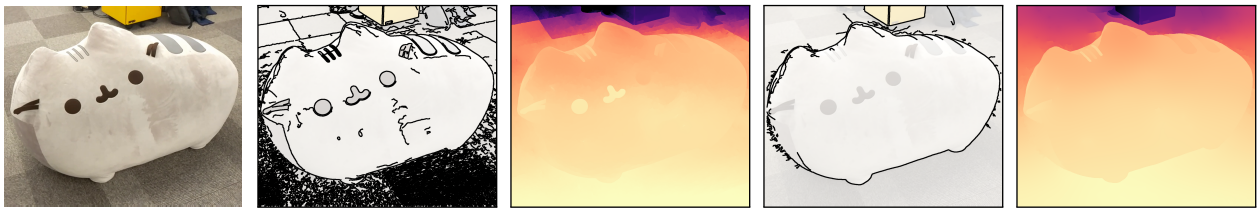
(a) Color image **(b)** Initial soft depth edges M_F **(c)** Filtered $\widetilde{M}_F$

Figure 3.16: Depth edge filtering. (b) Depth edges are well identified in the flow gradient image, but not well aligned with the image edges. (c) We apply spatio-temporal filtering to reduce noise and ensure the depth edges overlap with their corresponding image edges. Now they are ready for alignment with the image edges (see Figure 3.17).



(a) Image I **(b)** Magnitude of gradient $\widetilde{M}_I$ **(c)** Standard Canny using $\widetilde{M}_I$ **(d)** Magnitude of flow gradient $\widetilde{M}_D$ **(e)** Canny using $\widetilde{M}_I$ and $\widetilde{M}_D$

Figure 3.17: Localizing and aligning the soft depth edges. (a-b) Input image and corresponding magnitude of intensity gradient. (c) An edge detector that uses this image selects both texture and depth edges. Strong and weak edge pixels are drawn in bright and dark color, respectively. (d-e) We inject our soft edge map in the detector algorithm, which results in suppressed texture edges. The remaining edges are mostly depth edges and well aligned with the intensity edges (Note, that edges are drawn thick here for illustration purposes).



(a) Image I **(b)** Canny edges **(c)** Densification using (b) **(d)** Our predicted occlusion edges **(e)** Densification using (d)

Figure 3.18: Occlusion edges. The standard Canny edge detector fires on texture edges (b), which causes incorrect discontinuities in the resulting depth map (c). Our algorithm suppresses most texture edges (d), which leads to a refined depth map (e).

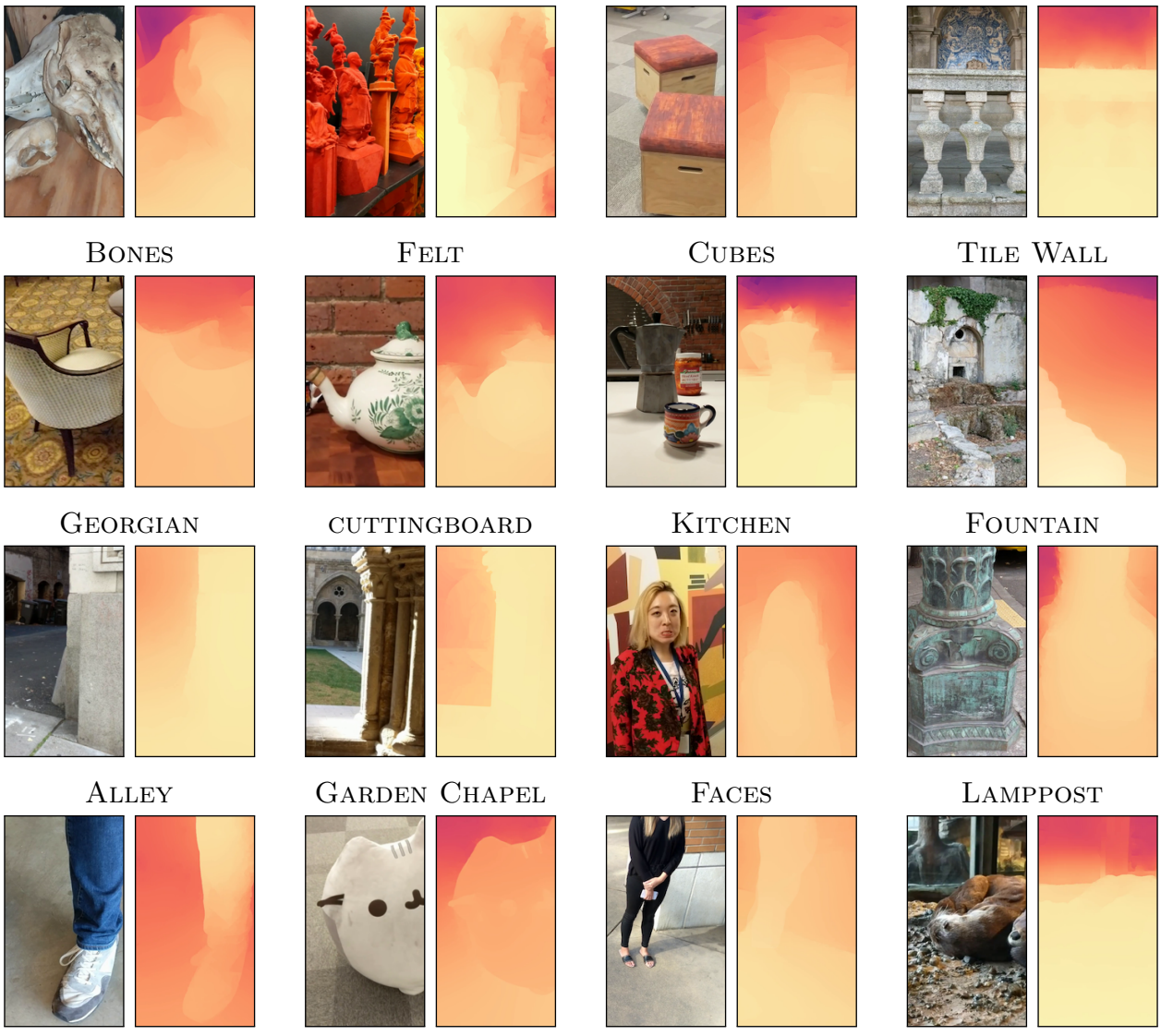
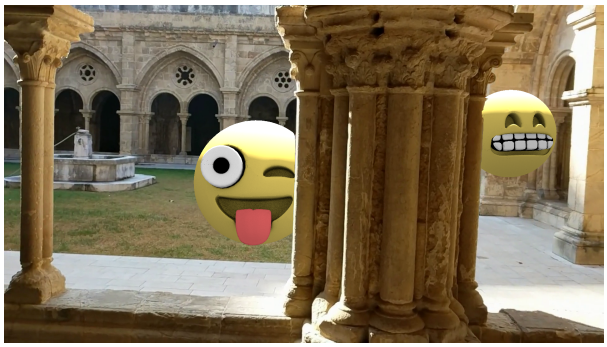


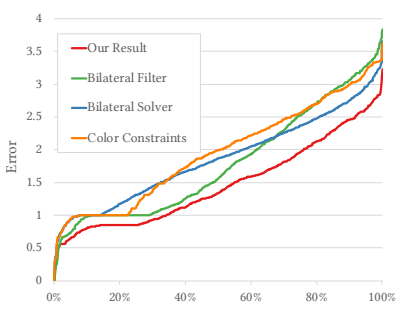
Figure 3.19: Datasets. Datasets we captured for this paper. Please refer to the online supplementary material for a full collection of video results. Note that images have been cropped for visualization. Our datasets consist of both portrait and landscape videos.



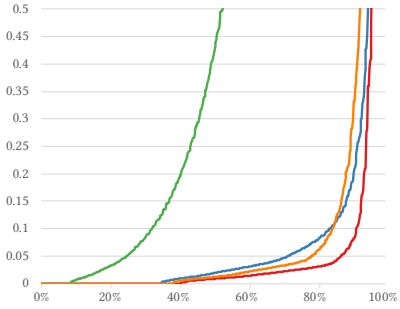
(a) Object insertion

(b) Virtual point light

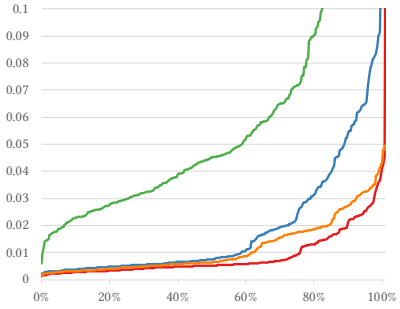
Figure 3.20: Visual effects. Example occlusion-aware AR effects.



(a) Occlusion error (Eq. 3.21)



(b) Texture error (Eq. 3.22)



(c) Temporal stability error (Eq. 3.23)

Figure 3.21: Quantitative comparison. Comparing our method against several baselines using our evaluation metrics. The graphs plot the cumulative error histogram for all annotated samples for five datasets (see text).

Chapter 4

TEMPORAL CONTEXT: EXTENDING IMAGES AND VIDEOS IN TIME

In this chapter, we describe our work on recovering temporal context, showing two examples: (1) automatically animating images into looping videos by injecting synthesized motion [64] and (2) a similar technique for increase the framerate or temporal extent of an existing video.

4.1 Single-image animation

For humans, a picture often contains much more than a collection of pixels. Drawing from our previous observations of the world, we can recognize objects, structure, and even imagine how the scene was moving when the picture was taken. Using these priors, we can often envision the image as if it were animated, with smoke billowing out of a chimney, or waves rippling across a lake. In this section, we propose a system that learns these same motion priors from videos of real scenes, enabling the synthesis of plausible motions for a novel static image and allowing us to render an animated video of the scene.

General scene motion is highly complex, involving perspective effects, occlusions, and transience. For the purposes of this section, we restrict our attention to fluid motions, such as smoke, water, and clouds, which are well approximated by Eulerian motion, in particular, particle motion through a static velocity field.

Our proposed method takes as input a single static image and produces a looping video texture. We begin by using an image-to-image translation network [180] to synthesize an Eulerian motion field. This network is trained using pairs of images and motion fields, which are extracted from a large collection of online stock footage videos of natural scenes.

Through Euler integration, this motion field defines each source pixel's trajectory through the output video sequence. Given the source pixel positions in a future frame, we render the corresponding frame using a deep warping technique: we use an encoder network to transform the input image into a deep feature map, warp those features using a novel *temporally symmetric* splatting technique, and use a decoder network to recover the corresponding warped color image. Lastly, in order to ensure our output video loops seamlessly, we apply a novel video looping technique that operates in deep feature space.

Our contributions include (1) a novel motion representation for single-frame textural animation that uses Euler integration to simulate motion, (2) a novel *symmetric* deep splatting technique for synthesizing realistic warped frames, and (3) a novel technique for seamless video looping of textural motion.

4.1.1 Previous Work

In addition to the prior work discussed in Chapter 2, and most similar to our work, Endo et al. [35] demonstrate high-quality motion and appearance synthesis for animating timelapses from static landscape imagery. Given an input image, this method predicts an incremental flow field, which is used to warp the input image. To generate a video, this process is repeated recurrently on the warped output frames. This approach theoretically enables generation of an infinite number of future frames at high-resolution. In practice, however, recurrent estimation often results in longer-term distortion. Here, we further highlight the differences between [35] and our proposed method:

1. [35] recurrently synthesizes a flow field per output frame. Instead, our method synthesizes a single motion field for the entire output sequence, and applies it recurrently as an Eulerian motion description. As a result, future motion predictions are not dependent on synthesized frames, and thus motion quality does not degrade over time. Our motion representation does assume that the motion is well approximated as fluid motion, but this assumption holds true for all the animations shown in [35, 98].

2. [35] estimates backward flow fields for inverse warping, whereas our method instead estimates a forward warp field and performs splatting. The latter approach is preferable for single-frame motion estimation, as the estimated motion field is spatially correlated with the input image, allowing the network to more easily predict sharp motion boundaries that are aligned with object boundaries in the image. This detail is particularly important for animating objects which are partially occluded, as it allows us to clearly delineate which parts of the scene are moving, and which are not.

3. [35] animates a video by explicitly warping RGB pixels, whereas our method performs warping in the deep feature domain, and thus does not contain many of the same characteristic artifacts of warping, such as shearing or rubber-sheeting. [98] also performs image warping using a neural network, by optimizing a latent code to reproduce the input image, and then warping the dynamic component of that latent code to synthesize future frames. This method is based on the StyleGAN framework, thus it requires a very large dataset for training and is restricted to a single type of scene (such as landscape photographs). Additionally, this method does not synthesize motion, and instead relies on random homography warps. This allows for animation of clouds (which can feasibly move in any direction), but does not extend to general scenes.

4. [35] creates looping videos by first generating an animated video and then crossfading between the ends of the generated video. This approach can produce convincing animations when the scene is sufficiently unstructured, the video is relatively long, and strong textures have been warped far from their starting positions. However, with increasing scene structure, shorter videos, or smaller motions, directly crossfading can result in significant ghosting or double-edge artifacts. We avoid these artifacts through a novel looping technique, which enforces a looping video in the deep feature domain.

In our evaluations, we provide comparisons to this technique, showing that our method more reliably estimates motion for scenes with fluids and animates videos with fewer visible

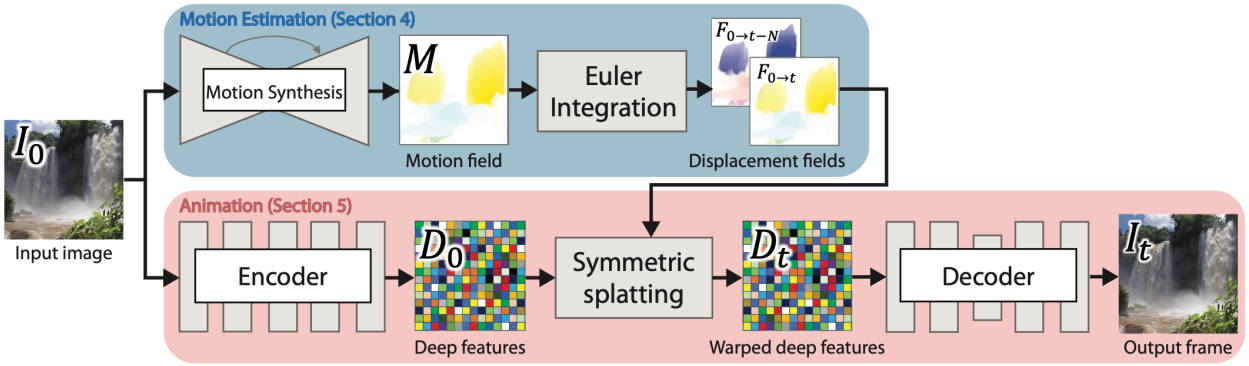


Figure 4.1: Overview. Given an input image I_0, our motion estimation network predicts a motion field M. Through Euler integration, M is used to generate future and past displacement fields $F_{0\to t}$ and $F_{0\to t-N}$, which define the source pixel locations in all other frames t. To animate the input image using our estimated motion, we first use a feature encoder network to encode the image as a feature map D_0. This feature map is warped by the displacement fields (using a novel symmetric splatting technique) to produce the corresponding warped feature map D_t. The warped features are provided to the decoder network to create the output video frame I_t.

artifacts.

4.1.2 Overview

Given a single static image I_0, we generate a looping video of length $N + 1$, consisting of frames I_t with $t \in [0, N]$. Our pipeline begins by using an image-to-image translation network to estimate a corresponding motion field M (Section 4.1.3), which is used to define the position of each pixel in all future frames. We use this information to animate the image through a deep warping technique (Section 4.1.4). Finally, in order to produce seamlessly looping videos, we introduce a technique to ensure that our videos always start and end with the same frame (Section 4.1.4). Our approach is summarized in Figure 4.1.

4.1.3 Motion estimation

We begin by describing the motion model and the motion estimation network. Given an image as input, we wish to synthesize plausible motion for the observed scene. Prior work

accomplishes this task through recurrent prediction of incremental flow fields [35], theoretically enabling generation of an infinite number of future frames at high-resolution. In practice, however, recurrent estimation often results in long-term distortion, since predicted motions are dependent on previously generated frames. In contrast, our motion field is only predicted once, given the input image, and thus does not degrade over time. Even though we use a single static motion field to represent the motion of an entire video, we can still model complex motion paths. This is because our motion field M is a static *Eulerian* flow field, i.e., a 2D map of motion vectors where each pixel's value defines its immediate velocity, which does not change over time. We use M to simulate the motion of a point (particle) from one frame to the next via Euler integration:

$$\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_t + M(\hat{\mathbf{x}}_t), \tag{4.1}$$

where $\hat{\mathbf{x}}_t$ is the point's (x, y) coordinate in frame t. In other words, treating each pixel as a particle, this motion field is the flow between each frame and its adjacent future frame:

$$M(\hat{\mathbf{x}}_t) = F_{t \to t+1}(\hat{\mathbf{x}}_t) \tag{4.2}$$

To synthesize this motion field, we train an image-to-image translation network [180] on color-motion pairs, such that when provided with a new color image I_0, it estimates a plausible motion field M. Given an image, M is only estimated once through an inference call to the network. Once estimated, it can be used to define the source pixel positions in all future frames t by recursively applying:

$$F_{0 \to t}(\hat{\mathbf{x}}_0) = F_{0 \to t-1}(\hat{\mathbf{x}}_0) + M(\hat{\mathbf{x}}_0 + F_{0 \to t-1}(\hat{\mathbf{x}}_0)) \tag{4.3}$$

This results in displacement fields $F_{0 \to t}$, which define the trajectory of each source pixel in I_0 across future frames I_t. These displacement fields are then used for warping the input image, as further described in Section 4.1.4. Computing $F_{0 \to t}$ does not incur additional calls to the network — it only uses information from the already-estimated M.

Note that unlike Endo et al. [35], who predict backward flow fields for warping (i.e., using bilinear backward sampling), we predict the *forward* motion field, i.e., aligned with

the input image. In our evaluations, we show that predicting forward motion results in more reliable motion prediction and sharper motion estimates at object boundaries. As a result, this enables more realistic animation of scenes with partial occlusions, since regions that are moving are more precisely delineated from those that are not.

4.1.4 Animation

Once we have estimated the displacement fields $F_{0\to t}$ from the input image to all future frames, we use this information to animate the image. Typically, forward warping, i.e., warping an image with a pixel-aligned displacement field, is accomplished through a process known as splatting. This process involves sampling each pixel in the input image, computing its destination coordinate as its initial position plus displacement, and finally assigning the source pixel's value to the destination coordinate. Warping an image with splatting unfortunately suffers from two significant artifacts: (1) the output is seldom dense — it usually contains holes, which are regions to which no source pixel is displaced, and (2) multiple source pixels may map to the same destination pixel, resulting in loss of detail or aliasing. Additionally, the predicted motion fields may be imperfect, and naively warping the input image can result in boundary artifacts. In the following section, we introduce a deep image warping approach to resolve these issues.

Deep image warping Given an image I_0 and a displacement field $F_{0\to t}$, we adopt a deep warping technique to realistically warp the input frame and fill unknown regions. Our method consists of three steps: (1) use an encoder network to encode the input image I_0 as a deep feature map D_0, (2) use the estimated displacement field $F_{0\to t}$ to splat those features to a future frame, producing D_t, and (3) use a decoder network to convert the warped features to an output color image I_t. For our encoder and decoder networks, we use variants of the architectures proposed in SynSin [183]. More implementation details are provided in Section 4.1.5.

As mentioned in the previous section, unlike backward warping, splatting may result

in multiple source pixels mapping to the same destination coordinate. In these cases, it is necessary to decide which value will occupy the pixel in the destination image. For this, we adopt softmax splatting [115], which assigns a per-pixel weighting metric Z to the source image, and uses a softmax to determine the contributions of colliding source pixels in the destination frame:

$$D_t(\hat{\mathbf{x}}') = \frac{\sum_{\hat{\mathbf{x}} \in \mathcal{X}} D_0(\hat{\mathbf{x}}) \cdot \exp(Z(\hat{\mathbf{x}}))}{\sum_{\hat{\mathbf{x}} \in \mathcal{X}} \exp(Z(\hat{\mathbf{x}}))} \tag{4.4}$$

where $\mathcal{X}$ is the set of pixels which map to destination pixel $\hat{\mathbf{x}}'$. Our method infers Z automatically as an additional channel of the encoded feature map. The learned metric allows the network to assign importance to certain features over others, and the softmax exponentiation avoids uniform blending, resulting in sharper synthesized frames.

Symmetric Splatting. As feature pixels are warped through repeated integration of our motion field M, we typically observe increasingly large unknown regions (Figure 4.2), occurring when pixels vacate their original locations and are not replaced by others. This effect is especially prominent at motion "sources", such as the top of a waterfall, where all predicted motion is outgoing. Although our decoder network is intended to fill these holes, it is still desirable to limit the complexity of the spatio-temporal inpainting task, as asking the network to animate an entire waterfall from a small set of distant features is unlikely to produce a compelling and temporally stable video.

Our solution to this problem leverages the fact that our motion is *textural* and *fluid*, and thus much of the missing textural information in unknown regions can be feasibly borrowed from other parts of the frame that lie along the same motion path. With this intuition in mind, we describe a *symmetric* splatting technique which uses reversed motion to provide valid textural information for regions which would otherwise be unknown.

So far, the process we have described to generate an animated video involves warping the encoded feature map D_0 by $F_{0 \to t}$ to produce future feature maps $V_f = \{D_0 ... D_N\}$, which are decoded to produce the output video frames. However, since our motion map M defines the motion between adjacent frames, we could just as easily animate the image by generating

Figure 4.2: Deep warping. Above: Naïve splatting of RGB pixels results in increasingly large unknown regions over time, shown in magenta. Below: For the same frames, our deep warping approach synthesizes realistic texture in these unknown regions.

a video of the past, i.e., instead of warping D_0 into the future, use $-M$ to compute $F_{0\to -t}$, resulting in warped feature maps $V_p = \{D_{-N}...D_0\}$. Decoding this feature video produces an equally plausible animation of the frame, with the main difference being that the large unknown regions in V_p occur at the start of the sequence, as opposed to at the end of the sequence in V_f.

In fact, because the direction of motion has been reversed, the motion sources have been replaced with motion "sinks" and vice versa (Figure 4.3). This means that the locations of the unknown regions in V_p are also largely complementary to those found in V_f. For instance, if our input image contains a waterfall, V_f will begin with the input feature map D_0, and pixels will gradually flow down the waterfall, eventually accumulating at the bottom, and leaving a large unoccupied region at the top. Conversely, V_p will begin with pixels accumulated at the top of the waterfall, and a large hole at the bottom, and will end with D_0. We leverage this complementarity by compositing pairs of feature maps (one in the past, one in the future) to produce a feature map which is typically fully dense.

We perform this composition through joint splatting: we splat each pixel of D_0 twice to the same destination frame, once using $F_{0\to t}$ and once using $F_{0\to t-N}$. Note that $F_{0\to t}$ does not necessarily equal $-F_{0\to -t}$, rather $F_{0\to -t}$ is the result of applying $-M$ recursively through Eq. 4.3. As before, we use the softmax splatting approach with a network-predicted per-pixel weighting metric to resolve conflicts. This process results in a composite feature map that seldom contains significant holes, enabling generation of longer videos with larger magnitude motion.

Looping In this section, we focus on ensuring that our output videos loop seamlessly. To this end, we first describe a modification to the splatting weights that guarantees that the first and last output video frames will be identical. Then, we describe an approach that enables end-to-end training without requiring a dataset of looping videos.

Prior work [35] produces looping videos through crossfading: an animated, but non-looping, video is generated first, and a crossfade is applied across the two ends of the video to smooth out any jarring frame transitions. This approach can be quite effective in certain

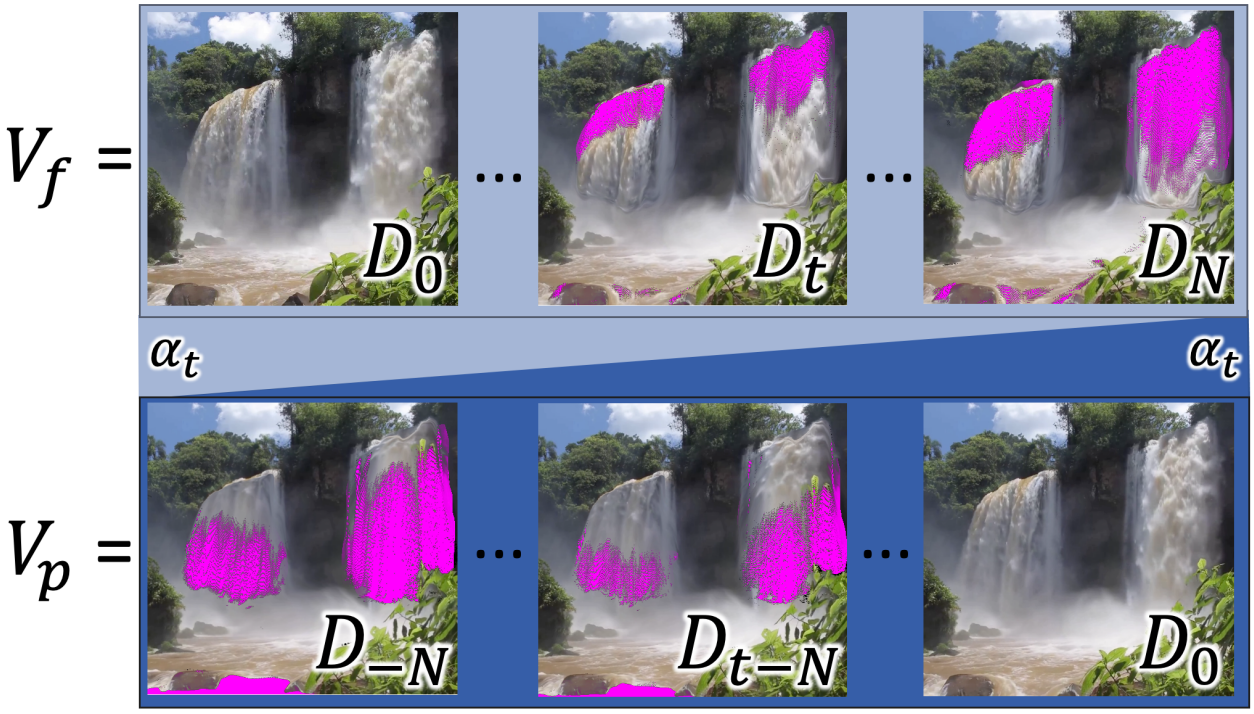


Figure 4.3: Seamless looping. An illustrated example of how seamless loops are created. Two feature videos are created by warping D_0. The first, V_f, contains the result of integrating the motion field M, resulting in a video starting with the input image and animating into the future. The second, V_p, instead uses $-M$, resulting in a video starting in the past and ending with the input frame. These two videos typically contain complementary unknown regions (shown in magenta). Before decoding, we combine the two feature maps via joint splatting. We modulate the contribution of each using splatting weights α_t, such that in the blended composite, the first and last frames are guaranteed to equal D_0, thus ensuring a seamless loop. Note that RGB images are shown for visualization, but these are both deep feature videos.

cases, but often produces artifacts in the form of double edges and ghosting. Instead of directly crossfading the animated video, our approach performs the transition in deep feature space, and provides the smoothly transitioning feature maps to the decoder. This allows us to enforce smooth transitions, while still producing images that contain realistic texture, avoiding many of the artifacts of direct crossfading.

Looping weights. Our looping technique relies on the observation that our two warped sequences V_p and V_f each have the input feature map D_0 on opposite ends of the sequence, as illustrated in Figure 4.3. With this in mind, if we are able to smoothly control the contribution of each, such that the first frame contains only the values in V_f and the last frame contains only the values in V_p, our feature maps (and our decoded images) on opposite ends of the video are guaranteed to be identical, and thus, our video is guaranteed to loop seamlessly. As such, we modulate the contribution of each feature map by introducing a temporal scaling coefficient to Eq. (4.4):

$$D_t(\hat{\mathbf{x}}') = \frac{\sum_{\hat{\mathbf{x}}\in\mathcal{X}} \alpha_t(\hat{\mathbf{x}}) \cdot D_0(\hat{\mathbf{x}}) \cdot \exp(Z(\hat{\mathbf{x}}))}{\sum_{\hat{\mathbf{x}}\in\mathcal{X}} \alpha_t(\hat{\mathbf{x}}) \cdot \exp(Z(\hat{\mathbf{x}}))} \tag{4.5}$$

where $\mathcal{X}$ is the set of pixels which map to destination pixel $\hat{\mathbf{x}}'$, either by warping forward or backward in time. For a given frame t, we set:

$$\alpha_t(\hat{\mathbf{x}}) = \begin{cases} \frac{t}{N} & \hat{\mathbf{x}} \in V_p \\ 1 - \frac{t}{N} & \hat{\mathbf{x}} \in V_f \end{cases} \tag{4.6}$$

Although the scaling coefficient α_t is linearly interpolated, the resulting composited feature video is not a linear interpolation of V_p and V_f, since coinciding splatted features from each are typically not from the same input locations, and thus have different values of Z. Since the value of Z is unconstrained and exponentiated, the overall magnitude of our weighting function ($\alpha_t(\hat{\mathbf{x}}) \cdot \exp(Z(\hat{\mathbf{x}}))$) can vary significantly, and thus our composited feature map seldom contains equally blended features. The added coefficient α_t serves as a forcing function to ensure that the composited feature maps D_t are equal to D_0 at $t = 0$ and $t = N$, but composited features will transition from V_f to V_p at different rates per-pixel, depending on the relative magnitudes of the splatted Z values.

Training on regular videos. Training our deep warping component (i.e., our encoder and decoder networks) to produce looping videos introduces an additional challenge: our training dataset consists of natural *non-looping* videos. In other words, the looping video we are tasking our networks with generating does not exist, even for our training examples, and thus, it's non-trivial to formulate a reconstruction loss for supervision. Therefore, as illustrated in Figure 4.4, we modify the task for training: instead of warping one frame in two directions, we use two different frames, one from the start of the video clip I_0^{GT}, and one from the end I_N^{GT}, encoded separately as feature maps. We additionally predict a motion field M from I_0^{GT}, which is integrated to produce displacement fields $F_{0\to t}$ and $F_{0\to t-N}$. The two feature maps, D_0 and D_N, are respectively warped by $F_{0\to t}$ and $F_{0\to t-N}$ to an intermediate frame t, using our joint splatting technique with the weights defined in Eq. 4.5. Finally, the composited feature map D_t is decoded to an image I_t, and a loss is computed against the real intermediate frame I_t^{GT}. At testing time, we perform the same process, except that instead of two input images, we use only one image, warped in both directions. This process is effectively training the network to perform video interpolation, and at inference time, using the network to interpolate between a frame and itself, while strictly enforcing the desired motion by warping the feature maps.

4.1.5 Implementation Details

In this section, we provide more details about the implementation of our method. First, we provide a summary of the network architectures used for the motion estimation and warping networks. Then, we provide details about our training and inference pipelines.

Network architecture. For the feature encoder and decoder networks, we use the architectures proposed in SynSin [183], which have shown compelling results for single-image novel-view synthesis. Since our aim is not to generate new viewpoints, but rather to animate the scene, we replace the reprojection component with the softmax splatting technique proposed in Niklaus et al. [115]. Additionally, we replace the noise-injected batch normalization

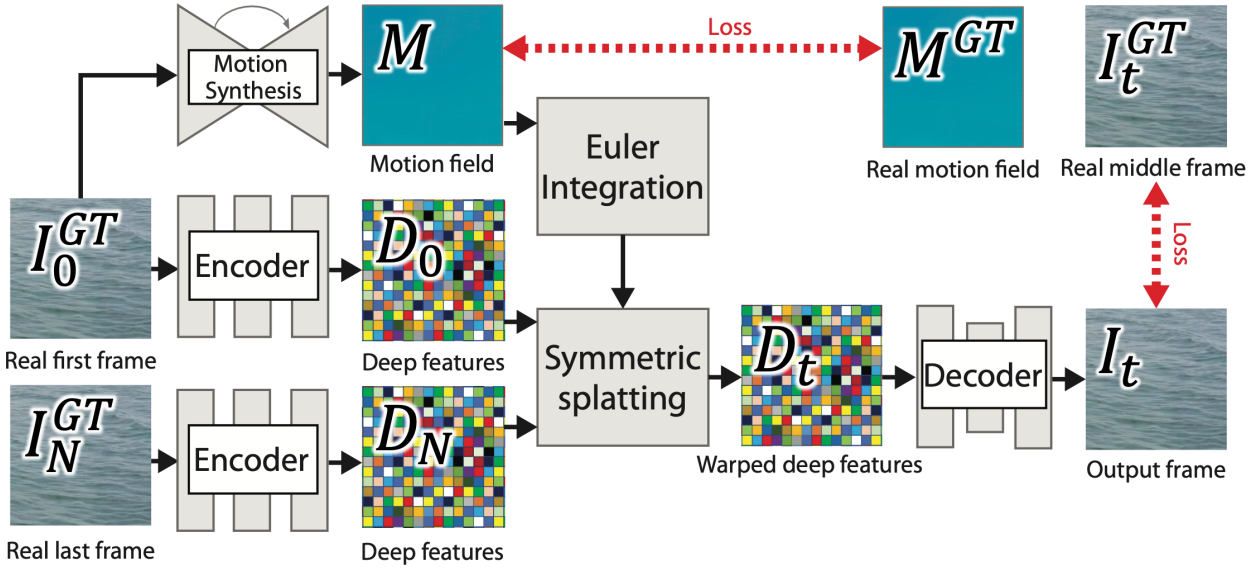


Figure 4.4: Training. As described in Section 4.1.4, each frame in our generated looping video is composed of textures from two warped frames. To supervise this process during training, i.e., to have a real frame to compare against, we perform our symmetric splatting using the features from two different frames, I_0 and I_N (instead of I_0 twice, as in inference). We enforce the motion field M to match the motion estimated from the ground truth video M^{GT}, and the output frame I_t to match the real video frame I_t^{GT}. For both, we use a combination of photometric and discriminative losses.

layer from SynSin with the modulated convolution approach proposed in Karras et al. [77] (to which we also provide a latent noise vector). This modification greatly helps reduce visual artifacts and enables stable discriminator training with smaller batch sizes (a necessity for limited GPU memory). For our motion estimation network, we use the architecture proposed in Pix2PixHD [180].

Training. We focus on natural scenes with fluid textures such as waterfalls, turbulent streams, and flowing waves. For our training data, we collected and processed a set of 1196 unique videos of textural motion from an online stock footage website[1]. We use 1096 for training, 50 for validation, and 50 for testing. To generate ground-truth motion fields, we

[1] www.storyblocks.com

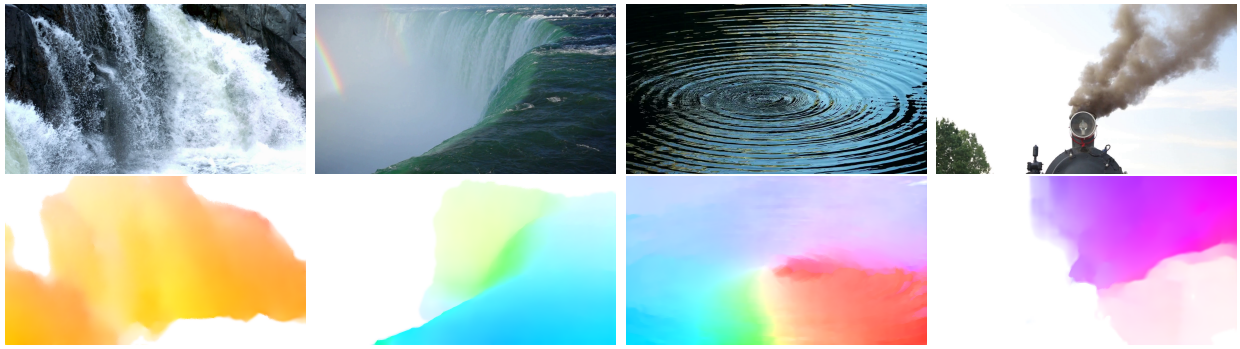


Figure 4.5: Data. Examples of the input images (top), alongside their corresponding synthesized motion fields (bottom). Full resolution images, along with their corresponding animated videos, can be found in the supplementary video.

use a pre-trained optical flow estimator [162] to compute the average optical flow between adjacent frames over a 2-second window. This effectively filters most motion which is cyclic, since pixels with cyclic motion will usually have been observed moving in opposing directions. We use only training videos from stationary cameras.

The motion estimation network is trained using 5 image-motion pairs from each of our 1096 training videos (a total of 5480 pairs) for 35 epochs, using the default parameters from Pix2PixHD [180]. Prior to training, we resize all our images to a standard size of 1280×720.

The warping component is trained on 5 short video clips from each of our 1096 training videos. A training triplet (start frame, middle frame, end frame) is selected from each video clip at random during training, further increasing the effective dataset size. We also apply random augmentation to our training examples, including horizontal flips and cropping. We train the network on batches of 8 images of size 256×256 for 200 epochs, using a discriminator learning rate of 3.5×10^{-3} and generator learning rate of 3.5×10^{-5}. We use the same losses and loss balancing coefficients shown in SynSin [183]. More details on the training schedule are provided in the supplementary material.

Inference. Our looping output videos have length $N = 200$ with a framerate of 30 frames per second. Each sequence is processed in 40 seconds on a Titan Xp GPU.

4.1.6 Results & Evaluation

We first present a quantitative analysis of our method, and show comparisons with the state-of-the-art in still-image animation [35] (Section 4.1.6), as well as ablated variations of our method. Then, we show qualitative results of our method on a diverse collection of input images (Section 4.1.6). We refer readers to our supplementary video for a full collection of visual results.

Quantitative evaluation In this section, we present our experiments evaluating the different components of our method, i.e., (1) a novel motion representation, (2) a novel symmetric splatting technique, and (3) a novel looping technique.

Motion representation. We evaluate the effectiveness of our proposed motion representation (integrated Eulerian flow) by comparing our predicted motion to ground truth pixel positions in future frames of the video. We establish ground truth motion by densely tracking all pixels through a sequence of 60 frames, using an off-the-shelf optical flow estimator [162]. We report the average Euclidean error between the ground truth positions and those estimated through our synthesized motion field, i.e., the endpoint error. We compare our proposed method to the following variants: (1) the per-frame recurrent estimation from Endo et al. [35], (2) directly predicting $F_{0\to N}$ and linearly interpolating intermediate motion $F_{0\to t}$ as $\frac{t}{N}F_{0\to N}$, and (3) training our motion network to predict the backward flow field, i.e., $M = F_{1\to 0}$ (and thus all splatting is replaced by backward warping). The results of this experiment can be found in Figure 4.6. We see that our method is able to most faithfully reproduce the ground-truth motion for our scenes. Empirically, we observe that the methods employing backward warping produce a majority of errors at motion boundaries, such as occlusions. We hypothesize that these differences are because the network is more easily able to predict an output that is spatially aligned with the input image.

Since images may often have many plausible motion directions, we ensure that the comparisons performed in this experiment are on video examples that contain unambiguous motion, eg. waterfalls and rivers. In order to identify these samples, we asked 5 users to

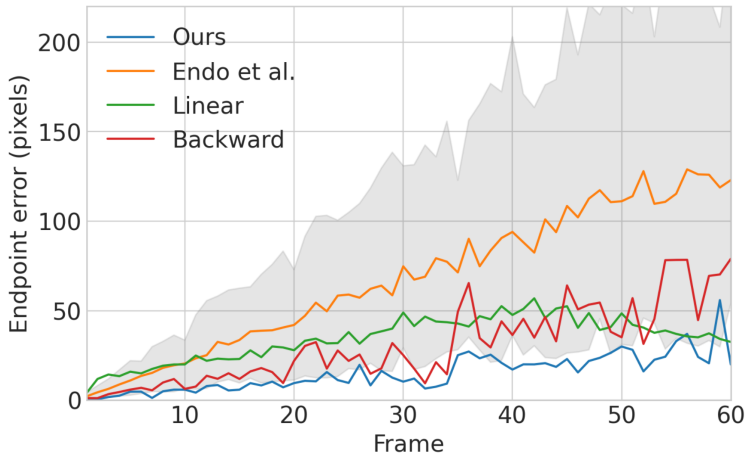


Figure 4.6: Quantitative evaluation: motion prediction. We evaluate the quality of the predicted motion by comparing the pixel positions in 60 future frames to those in the ground-truth video. We compare our proposed motion representation to three alternative methods, described in Section 4.1.6. The shaded region shows the range of predictions produced by Endo et al. [35]. We find that our proposed motion representation is able to most reliably reproduce the true motion information for scenes with fluid motion. All comparisons are performed on images of size 1280×720.

manually annotate the likely motion direction in 50 different images, and retained for quantitative comparison only the scenes in which all the annotations were within 30 degrees of the median ground truth motion direction. This results in a total of 32 clips, which we use for calculation of the motion and synthesis quantitative scores. Additionally, since we prefer the motion comparison to be agnostic to animation speed, i.e., animating the scene realistically but in slow-motion is acceptable, we solve for a per-sequence time-scaling constant that best aligns the motion magnitudes of the predicted and ground-truth displacement fields. This constant is computed for all methods and is used in all our comparisons.

The motion estimation network from Endo et al. [35] uses a latent code as input to the network, and different latent codes produce different predicted motions. To consider all possible outcomes of their method, we randomly sample 100 latent codes from the training codebook and report statistics on the resulting synthesized motions.

	PSNR↑	SSIM↑	LPIPS↓
Naïve color splatting	7.90	0.313	0.595
Backward Warping	10.29	0.409	0.483
Ours - $Z(\hat{\mathbf{x}}) = 1$	13.88	0.541	0.344
Ours - No Symmetric Splatting	12.19	0.493	0.418
Ours - Full	**14.63**	**0.619**	**0.313**

Table 4.1: Quantitative evaluation: video synthesis. We evaluate the quality of future frame predictions by comparing 60 synthesized frames with corresponding frames in the ground truth video. We compare our method to four alternatives, described in Section 4.1.6. All variant use our proposed motion estimation network.

Video synthesis. Second, we evaluate the choice of warping technique. Given the same flow values for a set of testing (unseen) video clips, we evaluate five future frame synthesis techniques: (1) naïve color splatting, where the feature encoder and decoder are not used, and instead the color values are warped, (2) backward warping, where the forward displacement field is inverted [145], and then backward warping is applied, such that no holes occur during warping, (3) our method without the network inferred weights, i.e., $Z(\hat{\mathbf{x}}) = 1$ for all pixels, (4) our method without symmetric splatting, and (5) our full method. We again use temporally-scaled sequences with unambiguous motion to compare each method's synthesized frames with the ground truth future video frames. We perform this comparison using PSNR, SSIM, and LPIPS [198]. Table 4.1 shows a quantitative comparison of these techniques, demonstrating that our proposed approach outperforms the alternatives at synthesizing future frames when the same motion is provided. Additionally, in the supplementary video, we show a qualitative comparison of these techniques. Compared to our approach, we observe that standard color splatting results in significant sparsity, i.e. many holes with unknown color. Backward warping instead fills these holes with interpolated (stretched) texture, which in most cases is equally jarring. Feature warping without inferred $Z(\hat{\mathbf{x}})$ values results in blurred details, since features are more often evenly combined. Remov-

	$S \geq 1$	$S \geq 2$	$S = 3$
Endo et al. [35]	348	101	9
Ours - No α_t	173	183	0
Ours - Crossfade	470	418	43
Ours - Full	**500**	**472**	**448**

Table 4.2: User study. We perform a user study to compare four techniques for producing looping videos. We collected 5 unique annotations for each of 100 samples. We direct users to judge the visual quality and realism of each looping video and rank the videos with unique scores $S = [0, 3]$, where 3 is best. We report the cumulative number of annotations above a certain ranking. On average, users rank our method higher than the alternatives.

ing symmetric splatting results in large unknown regions, which are filled in by the decoder network with blurry and often unrealistic texture.

Looping. Finally, we evaluate the choice of our looping technique. We compare four approaches: (1) our synthesis technique followed by the crossfading from Endo et al. [35], (2) the end-to-end pipeline described in Endo et al. [35], (3) our approach without the scaling coefficient α_t introduced in Eq. 4.5 and (4) our proposed approach. Since we do not have a ground truth looping video for comparison, we instead perform a user study, in which MTurk users are asked to rank the four variants by visual quality. This comparison is performed on 100 samples, consisting of two images sampled uniformly from each of the 50 testing sequences. Table 4.2 shows the results of the user study, which demonstrate that our proposed approach compares favorably against the alternatives. In the supplementary video, we show a visual comparison of these approaches. For our comparison to Endo et al. [35], we use the authors' implementation, trained on our dataset for the recommended 5000 epochs. Note that we do not use their appearance modulation component, as our scenes are not timelapses, and therefore do not have as significant changes in overall appearance.

Qualitative evaluation For evaluation purposes, we demonstrate our system on a large collection of still images. A subset of these images, along with their synthesized motions, can

be seen in Figure 4.5. The dataset contains a variety of natural scenes, including waterfalls, oceans, beaches, rivers, smoke, and clouds. In the supplementary video, we provide a larger set of input images and final rendered animations, as well as intermediate outputs such as synthesized motion fields.

In the results, we can see that the network learns important motion cues, such as perspective (i.e. motion is larger for objects closer to the camera), water turbulence, and detailed flow direction from surface ripples. By comparison, we find that the generated videos using the method in Endo et al. [35] more often produces videos with unrealistic motion or incorrect motion boundaries. Additionally, since our method performs warping in the deep feature domain, instead of explicitly warping RGB pixels, our results do not contain many of the same characteristic artifacts of warping, such as shearing or rubber-sheeting. Finally, we observe that our results loop more seamlessly, without obvious crossfading or ghosting.

Limitations. As mentioned in the introduction, our motion model targets *fluid* motion, and is therefore unsuitable for most cyclic motion, e.g. shaking trees. Additionally, our animations occasionally contain the following artifacts: (1) our motion estimation can fail to isolate thin occluding structures, and will animate them with their surroundings; (2) regions can be incorrectly identified as static, resulting in unnaturally frozen texture; (3) our warping technique does not model transparency — warping transparent surfaces results in animation of the refracted object; (4) classes of objects not seen in training are sometimes animated if they share similar textural properties to fluids. We provide examples of these failures in the supplementary video.

4.1.7 Conclusion

In this section, we have presented a method that can synthesize realistic motion from single photographs to produce animated looping videos. We introduced a novel motion representation for single-image textural animation that uses Euler integration. This motion is used to animate the input image through a novel symmetric splatting technique, in which we

combine texture from the future and past. Finally, we introduced a novel video looping technique for single-frame textural animation, allowing for seamless loops of our animated videos.

We demonstrated our method on a wide collection of images with fluid motion, and showed that our method is able to produce plausible motion and realistic animations.

4.2 The importance of accurate optical flow

As shown in the previous section, the first step in animating an image is to synthesize motion using a neural network that has learned motion priors from real videos. In the standard fully-supervised version of this pipeline, a neural network is shown a still frame from the video and tasked with synthesizing per-pixel motion that matches the ground-truth motion in the real video. Unfortunately, obtaining this ground-truth motion is not easy. For real videos, there exist no capture systems that that can precisely measure an arbitrary scene's true motion. As an alternative, one can manually annotate the scene motion in a video, but this is seldom a feasible choice at scale, since manual annotation is both tedious and imperfect.

Therefore, in order to produce motion estimates for training these types of single-image motion estimation networks, we typically use the motion fields produced by dense pixel tracking algorithms, otherwise known as optical flow algorithms. These optical flow methods are the backbone for many motion synthesis pipeline, since the motion they predict for the training videos are effectively treated as ground-truth. Unfortunately, this means that errors in the optical flow estimates will hamper the performance of the trained motion synthesis network, either by adding undesirable noise during training or by helping form incorrect motion associations. This phenomenon, combined with the fact that optical flow methods are very seldom entirely error-free, is likely the reason why motion synthesis methods have only ever been shown to operate successfully on limited domains. In this section, we explore this problem in depth: what is optical flow, how does it work, and what causes it to fail?

(a) First frame (b) Second frame (c) Estimated flow (d) Colormap

Figure 4.7: Optical flow: Optical flow is typically estimated between a pair of frames (a and b). The resulting optical flow field (c) describes the motion of all pixels from (a) to (b), i.e. each pixel contains a 2D motion vector describing that pixel's motion to the following frame. The optical flow field is shown as a color image, using the color mapping from [9], where different hues indicate motion in different directions, and the saturation indicates the relative magnitude. (d) shows a legend for this color mapping. Optical flow estimates made by state-of-the-art systems, in this case [162], are not always perfect. This can be seen in the optical flow field — the motion boundaries, seen as edges in (c), do not correspond with the object boundaries in the color image (a).

4.2.1 Background

Optical flow is the problem of estimating the two-dimensional motion of all pixels from one image to another. More specifically, an optical flow method will take as input two images (a source and destination image) and produce as output a *flow field*: an image where each pixel contains a 2D motion vector defining the displacement of a pixel's position from one frame to the next. An example of the inputs and output of optical flow estimation can be found in Figure 4.7.

Unlike sparse tracking methods, which aim at estimating the motion of some easily-tracked keypoints in the image, optical flow methods must produce motion estimates for *all pixels*. This makes the problem significantly more challenging, as the motion of many points will be ambiguous (Figure 4.8), such as textureless regions, or regions with changes

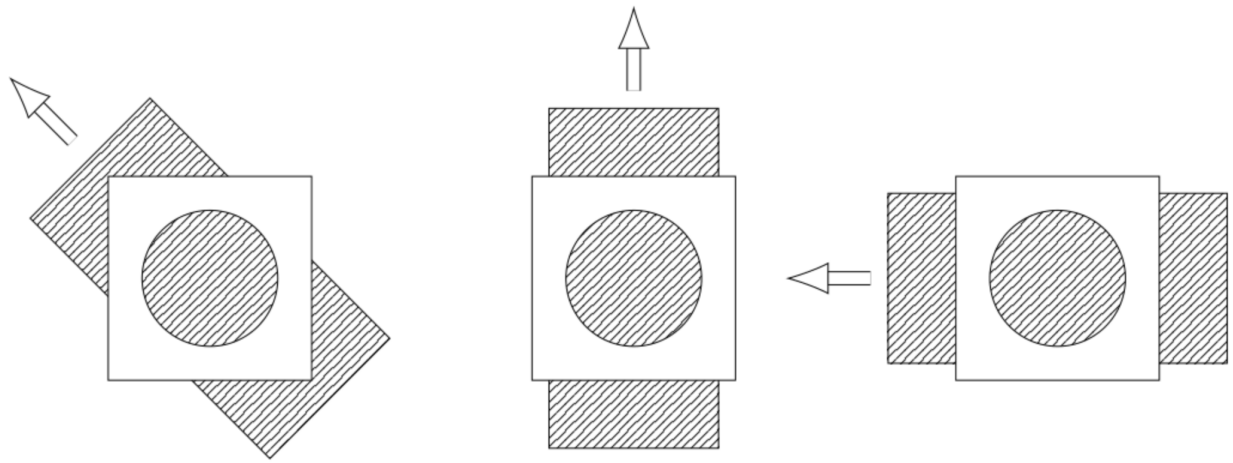


Figure 4.8: The aperture problem: In all three examples, the underlying object (a scored rectangle) is moving in different directions, yet the perceived motion in the captured image (the area within the circle) will be the same. This demonstrates a simple example of a type of motion ambiguity often found in real world scenes.

in lighting or appearance.

4.2.2 Existing methods

Optical flow estimation techniques typically operate through an optimization or energy minimization process: a metric is defined that measures the similarity of pixels across images, and an optimization method is used to maximize the similarity between the selected corresponding pixels. This metric can be relatively simple, like the color similarity of the two pixels, or more complex, like a comparison of windowed patches of pixels, image gradients [101], or even extracted feature descriptors such as SIFT [100, 96]. The optimization process also typically includes regularization [67], i.e., energy terms that encourage spatial smoothness so that nearby or visually similar pixels have similar motion estimates. These regularization terms can often help avoid local ambiguities caused by the aperture problem (Figure 4.8).

Historically, these methods have also relied on simplifying assumptions, such as *brightness*

constancy, the assumption that each point in the scene will stay the same color regardless of its position in the scene, or *small motion*, the assumption that frames come from a reasonably high-framerate video or from a slow-moving scene, and thus pixels will not have been displaced very far between any two frames. While these aid in constraining the problem, the assumptions do not typically hold in real-world scenarios: objects can move in and out of shade, can have arbitrarily fast or slow motion, and optical flow is not always estimated between adjacent video frames.

Recent works address these limitations through the use of deep learning. Among these, the most effective ones are *supervised* systems, meaning that they use ground-truth optical flow in the training process. In order to achieve this, these systems rely on synthetic datasets, such as Sintel [20] and FlyingChairs [34] (where true optical flow can be derived from the synthetic scene's 3D geometry), or carefully annotated real-world scenes, such as KITTI [45]. Deep learning-based optical flow estimation techniques can be broadly categorized into two groups: *feature-extraction networks* and *end-to-end systems*.

Feature extraction networks operate similarly to classical energy minimization techniques, but instead of comparing patches of pixels to determine their similarity in optimization, convolutional neural networks are used to extract deep latent features that represent local regions. These deep features are then used in traditional optimization frameworks, where optical flow values are chosen to minimize the feature distance between corresponding pixels. The use of a neural network for feature extraction lifts many of the previous assumptions of brightness constancy, since deep features can encode information about local patches in a way that is invariant to changes in lighting or appearance. In training, the neural network which operates as a feature extractor is trained to produce more meaningful (discriminative) features by backpropagating the error with the ground truth optical flow.

The second group of deep-learning based systems are end-to-end techniques. Instead of using a network to estimate features which are matched using classical energy-based techniques, a neural network can replace the entire process. The first techniques to implement this approach [34] struggled to match the accuracy of traditional energy-based techniques. A

number of methods improved on this baseline by incorporating architectural decisions that mimic techniques frequently used in classical optical flow estimation, such as coarse-to-fine pyramids [162], matching cost-volumes [162], and iterative refinement [69, 170].

One fundamental limitation of all these approaches is their difficulty in generalizing to scenes beyond the domains seen during training. Since these supervised techniques require ground-truth optical flow estimates for training, and there are a limited collection of datasets with ground truth optical flow (which happen to mostly be synthetic), the performance of these networks on real-world data seldom matches the performance on synthetic scenes. Supervised approaches have found ways of lessening, but not entirely removing, the effects of this *domain gap* through deliberate pre-training and fine-tuning schedules [71].

We can avoid this domain gap by training flow estimation networks in an *unsupervised* fashion. In this way, ground-truth optical flow is not required during training, and therefore networks can be trained on arbitrary data, including a large variety of real-world videos. In order to compute gradients for training, unsupervised methods use proxy losses, such as photometric consistency or perceptual similarity [198] between corresponding pixels. While unsupervised methods do not degrade as drastically on unseen domains, they typically perform worse than supervised methods, for the same reasons that traditional methods struggle with ambiguous motion.

A more comprehensive survey of effective optical flow estimation techniques can be found in the survey articles of Baker et al. [9], Hur and Roth [70], and Sun et al. [161].

4.2.3 Limitations for motion synthesis

In the previous section, we described a number of approaches for performing optical flow estimation and detailed the limitations of each. In addition to these limitations, there is one additional consideration when using these methods for the problem of motion synthesis.

In order to train a network to synthesize the motion necessary for converting an image into a video, we need more supervision than just a single optical flow field. In other words, in order to task our network with generating the motion of an entire video, the training

examples must also encode the motion of entire videos. Specifically, for each pixel, we need not only a single motion vector (as in Chapter 4.1 and the above described optical flow methods), but rather an entire motion *trajectory*. Unlike two-frame optical flow, this type of information, i.e., a dense collection of pixel trajectories over time, is not trivial to compute using standard methods. This is because optical flow methods are designed to estimate the motion between two distinct images, not an arbitrarily large collection of video frames. While there do exist a number of multi-frame (i.e., > 2 frame) optical flow techniques [79], these typically focus on small bundles of 3 or 5 frames, with the intention of improving flow estimates for occluded pixels, and do not extend to a larger collection of inputs.

In order to estimate a pixel's trajectory through a video using existing techniques, it is necessary to repeatedly evaluate a two-frame optical flow estimation algorithm on many pairs of frames from the video sequence. One approach is to compute the pixel's trajectory *sequentially*, i.e., by computing flow fields for frames $([0 \rightarrow 1], [1 \rightarrow 2], ..., [n-1 \rightarrow n])$, and chaining the flow estimates to produce the pixel's position in a particular frame. Alternatively, the optical flow can be computed *directly* between an arbitrary frame and all other frames, i.e., $([i \rightarrow 0], [i \rightarrow 1], ..., [i \rightarrow n])$, precluding the need for chained estimates. Unfortunately, both approaches have their own drawbacks. Directly computing the optical flow typically results in less accurate estimates, as optical flow estimation is typically less reliable over larger temporal offsets. This is because objects are harder to track under larger displacements, often due to changes in appearance over time. This decrease in accuracy usually manifests as a lack of temporal consistency, i.e., estimated positions flickering from frame to frame. Sequentially chained optical flow estimates, on the other hand, will very frequently result in drift, or the accumulation of errors, since any errors in a pairwise flow estimates will be carried over to all subsequent frames.

4.2.4 Full-video motion estimation

Instead of estimating optical flow between pairs of frames in the video, we can instead model the trajectory of each pixel as a continuous function:

$$p(x_0, y_0, t) => (x', y') \tag{4.7}$$

The function p models the collection of all pixel trajectories throughout the video, and is parameterized by the a canonical position of an pixel (x_0, y_0) (defined in an arbitrary frame t_0) and the query frame t. As output, it produces the pixel's position (x', y') at the queried frame. This function can be fit to a video by using self-supervised losses, e.g., heuristics such as color constancy (or even perceptual feature constancy), or through supervised losses, by encouraging the displacements between pairs of frames $p(x_0, y_0, t_i) - p(x_0, y_0, t_j)$ to match the output of an optical flow estimator between frames t_i and t_j.

Choosing the correct representation for this function is important. The function needs to be able to represent arbitrarily complex motion paths, but also needs to be reasonably smooth to avoid overfitting to noise in the supervisory signals. Coordinate-based neural networks, such as the multi-layer perceptron (MLP) networks recently popularized by NeRF [110] are a natural fit for this task. These networks encode spatial and temporal regularization through sinusoidal positional encoding, which can be used to selectively control the influence that each pixel has on its neighbors in space and time [10]. Despite this implicit regularization, these techniques are also able to represent arbitrarily high frequency details in the reconstructed signal, thanks to the adjustable frequency range of the positional encoding.

Evaluation To validate the effectiveness of this approach on real videos, we conduct an experiment on synthetic videos. In particular, we design a simplistic video sequence consisting of two layers with independent, semi-random motion, shown in Figure 4.9. We compare the quality of the motion estimates produced by our system with standard two-frame optical flow estimation techniques using the direct and chained variants described earlier in this section. To perform a more isolated experiment, and to minimize the effect of a domain gap, our synthetic sequence uses imagery with similar patch statistics to the Sintel [20] dataset,

Figure 4.9: Synthetic test sequence. The testing sequence used for evaluating long-term dense motion estimation. The video consists of 600 frames in which a textured disc rotates and bounces around the frame. The background also moves independently from frame-to-frame with a randomly initialized 2D Fourier series, resulting in a frame-to-frame motion of 5-20 pixels per frame.

and whenever relevant, we compare with models pre-trained on the Sintel dataset. Note that while we could use Sintel itself, this would preclude evaluation of long-term tracking, since Sintel sequences are typically relatively short, contain mostly transient content, and do not provide ground-truth motion estimates between non-adjacent frames.

As a quality metric, we use of endpoint error (EPE), the Euclidean distance between the estimated pixel position and the pixel's true position (computed analytically during generation of the synthetic sequence). We perform comparisons using two vartiants of this error:

(1) the mean endpoint error between the first frame and all later frames (i.e., $\{t_0 \rightarrow t_i, \forall i\}$). This metric should increase with overall error, including accumulated drift. We dub this metric EPE-D in Table 4.3.

(2) the mean endpoint error between consecutive frames (i.e. $\{t_i \rightarrow t_j, \forall i, j\}$). This error should increase with temporal inconsistencies, such as flickering position estimates, but should not be affected by drift. We dub this metric EPE-T in Table 4.3.

As can be seen in Table 4.3, our proposed video motion representation enables much higher quality motion estimates, especially over longer sequences.

	EPE-D	EPE-T
RAFT [170] (direct)	8.90	12.48
RAFT [170] (chained)	28.5	2.81
FlowNet2 [71] (direct)	9.14	14.8
FlowNet2 [71] (chained)	33.5	3.69
PWC-Net [162] (direct)	10.3	18.3
PWC-Net [162] (chained)	39.0	4.02
Ours	2.80	2.71

Table 4.3: Long-term motion estimation. We compare our proposed method with state-of-the-art two-frame optical flow estimation techniques. For all of the above experiments, we measure the endpoint error between the first frame of the video sequence and all future frames. The errors are computed from a source sequence with a duration of 600 frames and a resolution of 512x512.

4.2.5 Adding context to videos

The described technique for representing the motion of a video can be used for a number of applications that introduce additional context to our captured videos. In this section, we describe these potential applications, and demonstrate preliminary experimental findings that validate their feasibility.

Increasing temporal density Since the motion parameterization is continuous, we can sample the motion definition at any (fractional) point in time. As such, for all the pixels represented by this function, we can produce their positions at intermediate (i.e., unseen) frames. This information can be used to increase the temporal sampling rate of the video by advecting pixels from visible frames to novel, fractional timestamps. With this, we can produce higher framerate videos with smoother motion.

For small enough motions, simple color pixel advection may be sufficient, but for more complex or high-magnitude motions (in which holes may form between advected pixels), we can adopt similar approaches to the feature-based warping techniques shown in Section 4.1 and related works [115]. At the end of this section, we provide an evaluation of the quality of

motion estimates that can be expected from this technique. We find that, when compared to linear motion interpolation, the motion estimates using our trajectory function are usually significantly more accurate.

Extending videos The same technique can be applied to generate motion trajectories beyond the bounds of the video, i.e., extrapolating positions before or after the observed frames. In this scenario, the pixel positions sampled from the trajectory function are sampled from values of t outside the domain of the input video frames. Evaluating a function outside the supervised domain is generally unadvisable, but fortunately, the positional encoding used by our formulation makes our trajectory function periodic, and thus position values smoothly repeat beyond the bounds of the captured content. While these motion estimates are not necessarily accurate predictions of the future (since objects can move arbitrarily in the real unseen footage), the extrapolated periodic motion can often reproduce convincing animations for objects observed under complex periodic motion, such as tree leaves shaking in the wind.

Evaluation We measure the effectiveness of the above described interpolation approach using the same dataset as before, by holding out 10% of the frames, randomly dispersed throughout the video. As a metric, we once again compute the endpoint error between the ground truth pixel positions and the predicted positions in the above methods.

We compare our proposed approach with linear interpolation (as used in prior video interpolation literature [115]). For our synthetic test sequence, the mean endpoint error using our parametric motion interpolation is 0.31, compared to the endpoint error using linear interpolation, which is 0.89. While these are comparable at small baselines, if instead of holding out 60 randomly sampled frames, we hold out 6 randomly selected bundled of 10 frames each, we see a more significant difference, with a parametric EPE of 1.02 and a linear EPE of 9.11.

Chapter 5

DISCUSSIONS & CONCLUSION

5.1 Augmentation: synthesis or reconstruction?

In the context of augmenting memories, one must be careful to distinguish between *synthesized* and *reconstructed* information. These are often differentiated by the means through which they are recovered: information is typically referred to as synthesized if there is a heavy dependence on learned priors from other scenes, whereas reconstructed information is usually extracted strictly from the information provided in the scene. Most would likely refer to methods for recovering 3D structure (like those presented in Chapter 3) as reconstruction techniques, whereas video prediction methods (like those in Chapter 4) would likely be considered synthesis techniques.

In reality, this separation is not quite as clear. Strictly speaking, if a piece of information was not explicitly captured, we can seldom conclusively state that this information did indeed exist in the original scene. Most methods that are classified as reconstruction methods do perform some amount of data hallucination, whether they rely on learned priors or not. For example, 3D reconstruction techniques that do not use learned priors still define the likely shape of objects through the use of physically-based heuristics, like spatial smoothness. While these heuristics may hold true in a vast majority of cases, they may misrepresent a small collection of examples. Although misrepresenting the shape or motion of an old captured photo may not seem like a particularly serious consequence, one can imagine future uses of these technologies (beyond those presented in this thesis) in which contextual misrepresentation may have much more significant repercussions. One recent example is the collection of controversies that have resulted from the unequal representation of racial or ethnic groups in machine learning datasets. As a consequence, methods using this data are

often unable to operate effectively on certain groups of people, or even worse, will hallucinate wildly incorrect details that do not exist within a certain group.

With this in mind, as we continue to advance the state-of-the-art in synthesis and reconstruction, we must also consider how this generated content will be viewed. For instance, should augmented content be visibly classified as such? Should disclaimers be provided to identify this content as potentially hallucinated? Some may argue that it should, just as modern-day social media networks identify news articles from unverified sources.

These are pressing questions, as the line between reality and fiction has already begun to blur. The vast majority of images and selfies found online are the result of digital effects and manipulation through things like automatic face filters and Photoshop. A number of studies in recent years have even demonstrated the detrimental effects that doctored media has on our perception of the world [24]. With the advancement of technology, as we strive for more realistic visual effects, we can expect that many more adverse and unexpected consequences of our technologies will arise. In addition to this, we must also expect that these techniques will have the potential of being used for nefarious purposes. As such, it is of paramount importance that we begin developing techniques that are able to distinguish between real and synthesized imagery, even when our human eyes cannot. In recent years, we have seen the beginnings of this area of research [178], but as the synthesis technologies develop further, so must our techniques for identifying them.

5.2 Future-proofing our memories

Chapter 2 described the many forms of media that have emerged over the years, and detailed the contextual benefits and limitations of each. Additionally, it reviewed a number of approaches that recovered additional context from the limited data that is available.

Looking towards the future, it seems very likely that many of the modern technologies presented in this thesis will seem antiquated and contextually limited in another fifty years. This begs the question: as we begin to develop newer and more immersive technologies, what decisions might we be able to make today to facilitate the recovery of contextual data in the

future? A natural choice would be to simply capture and preserve as much information as possible, e.g., use multiple camera sensors, capture high resolution imagery, always capture a short video clip, or use a depth sensor. We can also consider other forms of sensory signals (i.e., not visual). Perhaps the next big breakthrough in realistically simulating our past memories and experiences is auditory, olfactory, gustatory, or tactile. While many of the technologies necessary for capturing and reproducing these senses do not yet exist, the information that they capture may be crucial in future applications.

5.3 Photography, realism, and artistic expression

This thesis focuses heavily on the importance of the photograph and other forms of media as mechanisms for capturing realistic portrayals of our experiences. One may argue that this interpretation is misguided, since our experiences are by definition subjective. As briefly mentioned in the introduction, our perception of the world is largely shaped by our culture and past experiences, and it is highly unlikely that multiple people will perceive an experience in precisely the same way (let alone a person and an algorithm). While I believe these arguments are valid for the narrative as a whole, I posit that the techniques presented in this thesis are aimed specifically at reconstructing *objective* phenomena: shape, position, object delineation, and camera movement.

It should also be acknowledged that beyond their use for capturing realistic depictions of experiences, photographs (as well as the other forms of media presented) are tools for creating art, i.e., content that evokes emotional response, but does not necessary hold correspondence to a physical memory. For this objective, one could argue that the addition of more and more realism (as is the trend in modern technology) is not necessarily beneficial, and that certain properties of what we may consider to be antiquated technologies are in fact complementary. An example of this is film grain: many movie directors still choose to capture their movies on film cameras as opposed to digital cameras because they appreciate the artistic effects of film.

Finally, we must also acknowledge that the importance of visual memories is not universal.

In this thesis, I argue that physical and digital media are important for preserving our visual experiences, such that we may continue to remember them in the future and pass them on to future generations. Not all cultures, however, engage with their history through visual means. In fact, some prefer less ocularcentric media, such as oral narratives, tactile artifacts, songs, or other representations. Many of these different representations also have different correspondence with the real world, as they do not represent a single moment or a single experience, but rather a sentiment, collection of events, or a shared experience.

5.4 Future work

The work presented in this thesis identifies a number of promising directions for future investigation.

Uses of full-video motion estimation A large collection of methods in the field of computer vision rely heavily on correspondences across multiple images. In most of these cases, these correspondences are extracted between frames through the use of two-frame optical flow techniques. As shown in Chapter 4.2, optical flow is often unreliable for longer videos, resulting in reduced performance of the end-goal applications. In order to improve these methods, instead of using optical flow to establish pixel trajectories over sequences of frames, we can adopt variants of our full-video motion estimation technique.

A natural application for this is video stabilization, which typically operates by estimating and compensating for a rigid transformation of the camera. Existing methods can fail if the estimated motion is incorrect, which occurs often in cases with low texture or other ambiguities between camera and object motion. One might expect that by using longer-term motion information, some of these ambiguities might be more easily resolved.

Another collection of use-cases are techniques that merge image-bursts. These techniques (which can perform high-dynamic range photography, denoising, or de-blurring, further discussed in Chapter 2) all rely on the spatial alignment of corresponding pixels across a burst of images. Spatial alignment tends to only be reliable over shorter bursts, since the cameras are often handheld and the motion after several frames is usually too significant to perform

reliable alignment. The use of more reliable full-video motion estimation may reduce some of these limitations, enabling integration of pixel values over longer captured sequences.

Aside from replacing optical flow methods with our video motion estimation approach, there exist a number of directions that are *only* possible through video-based motion estimation. As hinted in Chapter 4.2, these are methods that learn priors from long-term motion trajectories.

5.4.1 Learning from motion trajectories

Turning arbitrary images into videos In Chapter 4.1, we demonstrated an approach for using learned priors about how scenes move to animate fluid effects in still images. The obvious next step is to extend this approach to arbitrary motion. In order to enable this, it is necessary to solve several problems:

(1) A motion synthesis network must be trained with motion trajectories, such that it can synthesize an arbitrary pixel trajectory for a given novel image. The current pipeline relies on synthesizing a single motion field, which is only applicable to fluid motion animation. The type of motion that one may want to operate on, beyond fluid motion, may be periodic, linear, and stochastic, and as such the motion representation chosen for prediction must be able to reproduce these effects.

(2) After the motion is synthesized, animating an arbitrary image is also more challenging than with fluids, since arbitrary motion may contain multiple distinct layers that result in complex occlusions and disocclusions. While our feature warping approach may sometimes be able to simulate these effects, its ability to fill larger holes is limited. To enable more realistic synthesis of disocclusions, it is likely necessary to perform selective inpainting of each scene layer, as in recent work in novel-view synthesis [150].

One can also imagine smaller modifications to the current fluid motion synthesis pipeline that enable more realistic animations. One current effect that limits the realism of animated

videos is the use of repetitive (i.e., static) motion. As a result, stochastic effects like splashes are not simulated. To add stochasticity, we can additionally synthesize a per-pixel mean and variance for the magnitude and orientation of the motion vector. We can then sample a new vector from this distribution every time we generate a new frame of the output video.

Synthesizing motion in 3D Another future direction is the extension of our current work to three dimensions. One can imagine synthesizing motion in 3D, for instance by moving around particles in a NeRF [110]. Since supervision for object motion may not be available in 3D, we can instead contrain the synthesized motion in two dimensions, by projecting it to one of the captured viewpoints.

Disentangling camera and scene motion One major limitation of current techniques that learn priors from collections of videos is the fact that they require training videos to be captured with static cameras. This enables the extracted motion to correspond entirely to the content of the scene, as opposed to the motion of the camera. In practice, however, the vast majority of video sequences that exist online contain both scene and camera motion. Therefore, in order to expand the domain of our training data, it may be necessary to develop methods that are able to disentangle scene and camera motion. Techniques for performing this type of *scene flow* estimation exist already [108, 176], but seldom match the accuracy of 2D optical flow estimation techniques.

One potential solution is to extend our motion trajectory formulation to jointly estimate camera motion. However, there will likely still be ambiguous motions that can be reasonably explained by both camera or scene motion. In these cases, we may choose to learn priors about the disentanglement of camera and scene motion from large collections of captured (and pose-reconstructed) videos.

Seamlessly looping videos Finally, many of the same techniques described for video synthesis, interpolation, and extrapolation are particularly useful for the problem of creating looping videos. As briefly discussed in Chapter 2, video looping is a well-explored problem in computer vision, but existing methods have limitations in the type of content they are

able to loop. Current techniques are able to loop periodic or textual motion, i.e., videos that (at some point in time) see the same content repeated in the same spatial location. In many of our videos, however, we do not see this type of repetition, especially in shorter videos, handheld videos that change viewpoints, or when observing transient effects such as a person walking across the frame. In these cases, in order to create a seamless loop, it becomes necessary to synthesize additional motion that closes the gap between the endpoints of the video, enabling more seamless looping.

5.5 Conclusion

In this report, we explored the problem of legacy media augmentation. First, we began by identifying a number of key limitations of photographs that reduce the realism, immersion, and fidelity of our captured visual memories: photographs have fixed viewpoints, limited fields of view, and capture static single moments in time. We then presented a survey of modern forms of media, including panoramas, cinemagraphs, and augmented reality experiences. We also detailed existing methods for converting less immersive forms of media (like photographs) into more immersive ones (like 3D experiences), and presented our own progress towards two applications: (1) augmenting memories with spatial context by elevating videos to interactive 3D experiences, and (2) augmenting memories with temporal context, by turning images into animated looping videos and extending videos in time.

BIBLIOGRAPHY

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M Seitz, and Richard Szeliski. Building Rome in a day. In *International Conference on Computer Vision (ICCV)*, pages 72–79, 2009.

[2] Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin, and Richard Szeliski. Panoramic video textures. *ACM Transactions on Graphics (TOG)*, 24(3):821–827, 2005.

[3] 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 Transactions on Graphics (Proc. SIGGRAPH Asia)*, 35(6):article no. 198, 2016.

[4] Jason Antic. Deoldify: A deep learning based project for colorizing and restoring old images (and video!).

[5] Benjamin Attal, Selena Ling, Aaron Gokaslan, Christian Richardt, and James Tompkin. Matryodshka: Real-time 6dof video view synthesis using multi-sphere images. In *European Conference on Computer Vision*, pages 441–459. Springer, 2020.

[6] Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Selectively de-animating video. *ACM Trans. Graph.*, 31(4):66–1, 2012.

[7] Caroline Baillard, Cordelia Schmid, Andrew Zisserman, and Andrew Fitzgibbon. Automatic line matching and 3D reconstruction of buildings from multiple views. In *ISPRS Conference on Automatic Extraction of GIS Objects from Digital Imagery*, volume 32, pages 69–80, 1999.

[8] Simon Baker, Eric Bennett, Sing Bing Kang, and Richard Szeliski. Removing rolling shutter wobble. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2392–2399. IEEE, 2010.

[9] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.

[10] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *arXiv preprint arXiv:2103.13415*, 2021.

[11] Jonathan T Barron and Ben Poole. The fast bilateral solver. *European Conference on Computer Vision (ECCV)*, pages 617–632, 2016.

[12] Adrien Bartoli and Peter Sturm. Constrained structure and motion from multiple uncalibrated views of a piecewise planar scene. *International Journal of Computer Vision*, 52(1):45–64, 2003.

[13] Adrien Bartoli, Peter Sturm, and Radu Horaud. *A projective framework for structure and motion recovery from two views of a piecewise planar scene.* PhD thesis, INRIA, 2000.

[14] Kiran S Bhat, Steven M Seitz, Jessica K Hodgins, and Pradeep K Khosla. Flow-based video synthesis and editing. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 360–363. ACM, 2004.

[15] Brojeshwar Bhowmick, Suvam Patra, Avishek Chatterjee, Venu Madhav Govindu, and Subhashis Banerjee. Divide and conquer: Efficient large-scale structure from motion using graph partitioning. In *Asian Conference on Computer Vision (ACCV)*, pages 273–287, 2014.

[16] Nicolas Bonneel, James Tompkin, Kalyan Sunkavalli, Deqing Sun, Sylvain Paris, and Hanspeter Pfister. Blind video temporal consistency. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2015)*, 34(6), 2015.

[17] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[18] Michael Broxton, Jay Busch, Jason Dourgarian, Matthew DuVall, Daniel Erickson, Dan Evangelakos, John Flynn, Peter Hedman, Ryan Overbeck, Matt Whalen, et al. Deepview immersive light field video. In *ACM SIGGRAPH 2020 Immersive Pavilion*, pages 1–2. 2020.

[19] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, 2001.

[20] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European*

Conf. on Computer Vision (ECCV), Part IV, LNCS 7577, pages 611–625. Springer-Verlag, October 2012.

[21] Federico Camposeco and Marc Pollefeys. Using vanishing points to improve visual-inertial odometry. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 5219–5225. IEEE, 2015.

[22] John Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[23] Avishek Chatterjee and Venu Madhav Govindu. Efficient and robust large-scale rotation averaging. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 521–528, 2013.

[24] Jonlin Chen, Masaru Ishii, Kristin L Bater, Halley Darrach, David Liao, Pauline P Huynh, Isabel P Reh, Jason C Nellis, Anisha R Kumar, and Lisa E Ishii. Association between the use of social media and photograph editing applications, self-esteem, and cosmetic surgery acceptance. *JAMA facial plastic surgery*, 21(5):361–367, 2019.

[25] Chia-Chi Cheng, Hung-Yu Chen, and Wei-Chen Chiu. Time flies: Animating a still image with time-lapse video as reference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5641–5650, 2020.

[26] Yung-Yu Chuang, Dan B Goldman, Ke Colin Zheng, Brian Curless, David H Salesin, and Richard Szeliski. Animating pictures with stochastic motion textures. *ACM Transactions on Graphics (TOG)*, 24(3):853–860, 2005.

[27] Andrea Cohen, Torsten Sattler, and Marc Pollefeys. Merging the unmatchable: Stitching visually disconnected sfm models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2129–2137, 2015.

[28] 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 Transactions on Graphics (ToG)*, 34(4):1–13, 2015.

[29] Zhaopeng Cui and Ping Tan. Global structure-from-motion by similarity averaging. In *International Conference on Computer Vision (ICCV)*, pages 864–872, 2015.

[30] Brian Curless. From range scans to 3d models. *ACM SIGGRAPH Computer Graphics*, 33(4):38–41, 1999.

[31] Paul E Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *ACM SIGGRAPH 2008 classes*, pages 1–10. 2008.

[32] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry and image-based approach. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, volume 96, pages 11–20, 1996.

[33] P. Dollár, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *CVPR*, June 2006.

[34] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[35] Yuki Endo, Yoshihiro Kanamori, and Shigeru Kuriyama. Animating landscape: Self-supervised learning of decoupled motion and appearance for single-image video synthesis. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2019)*, 38(6):175:1–175:19, 2019.

[36] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018.

[37] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, pages 834–849, 2014.

[38] Olivier D Faugeras, Q-T Luong, and Stephen J Maybank. Camera self-calibration: Theory and experiments. In *European Conference on Computer Vision (ECCV)*, pages 321–334, 1992.

[39] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2367–2376, 2019.

[40] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2016.

[41] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik,

et al. Building Rome on a cloudless day. In *European Conference on Computer Vision (ECCV)*, pages 368–381, 2010.

[42] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81, 2015.

[43] Yasutaka Furukawa and Carlos Hernández. Multi-view stereo: A tutorial. *Foundations and Trends. in Computer Graphics and Vision*, 9(1-2):1–148, 2015.

[44] Ruohan Gao, Bo Xiong, and Kristen Grauman. Im2flow: Motion hallucination from static images for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5937–5947, 2018.

[45] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[46] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.

[47] Clément Godard, Kevin Matzen, and Matt Uyttendaele. Deep burst denoising. In *Proceedings of the European conference on computer vision (ECCV)*, pages 538–554, 2018.

[48] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, 1996.

[49] Venu Madhav Govindu. Combining two-view constraints for motion estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[50] Venu Madhav Govindu. Robustness in motion averaging. In *Asian Conference on Computer Vision (ACCV)*, pages 457–466, 2006.

[51] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: a Line Segment Detector. *Image Processing On Line*, 2:35–55, 2012.

[52] Matthias Grundmann, Vivek Kwatra, Daniel Castro, and Irfan Essa. Calibration-free rolling shutter removal. In *2012 IEEE international conference on computational photography (ICCP)*, pages 1–8. IEEE, 2012.

[53] Robert M Haralick, Hyonam Joo, Chung-Nan Lee, Xinhua Zhuang, Vinay G Vaidya, and Man Bae Kim. Pose estimation from corresponding point data. In *Machine Vision for Inspection and Measurement*, pages 1–84. Elsevier, 1989.

[54] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, 1988.

[55] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[56] Richard I Hartley. Lines and points in three views and the trifocal tensor. *International Journal of Computer Vision*, 22(2):125–140, 1997.

[57] Samuel W Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM Transactions on Graphics (ToG)*, 35(6):1–12, 2016.

[58] Peter Hedman. Free photography for virtual reality. In *Real VR–Immersive Digital Reality*, pages 132–166. Springer, 2020.

[59] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3D Photography. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 36(6):article no. 234, 2017.

[60] Peter Hedman and Johannes Kopf. Instant 3d photography. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.

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

[62] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM Transactions on Graphics (TOG)*, 35(6):1–11, 2016.

[63] Tobias Hinz, Matthew Fisher, Oliver Wang, and Stefan Wermter. Improved techniques for training single-image GANs. *arXiv preprint arXiv:2003.11512*, 2020.

[64] Aleksander Holynski, Brian L Curless, Steven M Seitz, and Richard Szeliski. Animating pictures with eulerian motion fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5810–5819, 2021.

[65] Aleksander Holynski, David Geraghty, Jan-Michael Frahm, Chris Sweeney, and Richard Szeliski. Reducing drift in structure from motion using extended features. In *2020 International Conference on 3D Vision (3DV)*, pages 51–60. IEEE, 2020.

[66] Aleksander Holynski and Johannes Kopf. Fast depth densification for occlusion-aware augmented reality. *ACM Transactions on Graphics (ToG)*, 37(6):1–11, 2018.

[67] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[68] Asmaa Hosni, Christoph Rhemann, Michael Bleyer, and Margrit Gelautz. Temporally consistent disparity and optical flow via efficient spatio-temporal filtering. In *Pacific-Rim Symposium on Image and Video Technology*, pages 165–177. Springer, 2011.

[69] Junhwa Hur and Stefan Roth. Iterative residual refinement for joint optical flow and occlusion estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5754–5763, 2019.

[70] Junhwa Hur and Stefan Roth. Optical flow estimation in the deep learning age. In *Modelling Human Motion*, pages 119–140. Springer, 2020.

[71] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[72] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, 2011.

[73] Wei-Cih Jhou and Wen-Huang Cheng. Animating still landscape photographs through cloud motion creation. *IEEE Transactions on Multimedia*, 18(1):4–13, 2015.

[74] Nianjuan Jiang, Zhaopeng Cui, and Ping Tan. A global linear method for camera pose registration. In *International Conference on Computer Vision (ICCV)*, pages 481–488, 2013.

[75] Andrew Jones, Ian McDowall, Hideshi Yamada, Mark Bolas, and Paul Debevec. Rendering for an interactive 360 light field display. In *ACM SIGGRAPH 2007 papers*, pages 40–es. 2007.

[76] Alexandre Karpenko, David Jacobs, Jongmin Baek, and Marc Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. *CSTR*, 1(2):13, 2011.

[77] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[78] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[79] Ryan Kennedy and Camillo J Taylor. Optical flow with geometric occlusion estimation and fusion of multiple frames. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 364–377. Springer, 2015.

[80] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. Robust consistent video depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1611–1621, 2021.

[81] Jana Kosecka and Wei Zhang. Video compass. In *European Conference on Computer Vision (ECCV)*, pages 476–490, 2002.

[82] Till Kroeger, Radu Timofte, Dengxin Dai, and Luc Van Gool. Fast optical flow using dense inverse search. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[83] Felicitas Lang and W Förstner. Surface reconstruction of man-made objects using polymorphic mid-level features and generic scene knowledge. *International Archives of Photogrammetry and Remote Sensing*, 31:415–420, 1996.

[84] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[85] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *ACM SIGGRAPH 2004 Papers*, pages 689–694. 2004.

[86] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Trans. Graph.*, 23(3):689–694, 2004.

[87] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996.

[88] Haoang Li, Jian Yao, Jean-Charles Bazin, Xiaohu Lu, Yazhou Xing, and Kang Liu. A monocular slam system leveraging structural regularity in manhattan world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2518–2525. IEEE, 2018.

[89] M. Li and A. I. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *International Journal of Robotics Research*, 32(6):690–711, May 2013.

[90] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Flow-grounded spatial-temporal video prediction from still images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 600–615, 2018.

[91] Chia-Kai Liang, Li-Wen Chang, and Homer H Chen. Analysis and compensation of rolling shutter effect. *IEEE Transactions on Image Processing*, 17(8):1323–1330, 2008.

[92] Jing Liao, Mark Finch, and Hugues Hoppe. Fast computation of seamless video loops. *ACM Transactions on Graphics (TOG)*, 34(6):197, 2015.

[93] Zicheng Liao, Neel Joshi, and Hugues Hoppe. Automated video looping with progressive dynamism. *ACM Transactions on Graphics (TOG)*, 32(4):77, 2013.

[94] Orly Liba, Kiran Murthy, Yun-Ta Tsai, Tim Brooks, Tianfan Xue, Nikhil Karnad, Qiurui He, Jonathan T Barron, Dillon Sharlet, Ryan Geiss, et al. Handheld mobile photography in very low light. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[95] Chih-Yang Lin, Yun-Wen Huang, and Timothy K Shih. Creating waterfall animation on a single image. *Multimedia Tools and Applications*, 78(6):6637–6653, 2019.

[96] Ce Liu, Jenny Yuen, Antonio Torralba, Josef Sivic, and William T Freeman. Sift flow: Dense correspondence across different scenes. In *European conference on computer vision*, pages 28–42. Springer, 2008.

[97] Haomin Liu, Guofeng Zhang, and Hujun Bao. Robust keyframe-based monocular slam for augmented reality. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 1–10. IEEE, 2016.

[98] Elizaveta Logacheva, Roman Suvorov, Oleg Khomenko, Anton Mashikhin, and Victor Lempitsky. DeepLandscape: Adversarial modeling of landscape video. *arXiv preprint arXiv:2008.09655*, 2020.

[99] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *arXiv preprint arXiv:1906.07751*, 2019.

[100] David G Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[101] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[102] Bicheng Luo, Feng Xu, Christian Richardt, and Jun-Hai Yong. Parallax360: Stereoscopic 360 scene representation for head-motion parallax. *IEEE transactions on Visualization and Computer Graphics*, 24(4):1545–1553, 2018.

[103] Xuan Luo, Jason Lawrence, and Steven M Seitz. Pepper's cone: An inexpensive do-it-yourself 3d display. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 623–633, 2017.

[104] Xuan Luo, Xuaner Zhang, Paul Yoo, Ricardo Martin-Brualla, Jason Lawrence, and Steven M Seitz. Time-travel rephotography. *ACM Transactions on Graphics (TOG)*, 40(6):1–12, 2021.

[105] D. Scaramuzza M. Pizzoli, C. Forster. REMODE: Probabilistic, monocular dense reconstruction in real time. *International Conference on Robotics and Automation (ICRA)*, pages 2609–2616, 2014.

[106] Ezio Malis and Manuel Vargas. Deeper understanding of the homography decomposition for vision-based control. Technical Report 6303, INRIA, 2007.

[107] James McCann and Nancy S Pollard. Real-time gradient-domain painting. In *ACM Transactions on Graphics (TOG)*, volume 27, page 93. ACM, 2008.

[108] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3061–3070, 2015.

[109] Branislav Micusik and Horst Wildenauer. Structure from motion with line segments under relaxed endpoint constraints. *International Journal of Computer Vision*, 124(1):65–79, 2017.

[110] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[111] Pierre Moulon, Pascal Monasse, and Renaud Marlet. Global fusion of relative motions for robust, accurate and scalable structure from motion. In *International Conference on Computer Vision (ICCV)*, pages 3248–3255, 2013.

[112] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[113] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. *International Conference on Computer Vision (ICCV)*, pages 2320–2327, 2011.

[114] Kai Ni, Drew Steedly, and Frank Dellaert. Out-of-core bundle adjustment for large-scale 3D reconstruction. In *International Conference on Computer Vision (ICCV)*, 2007.

[115] Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5437–5446, 2020.

[116] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[117] Yoshikuni Nomura, Li Zhang, and Shree K Nayar. Scene collages and flexible camera arrays. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 127–138, 2007.

[118] Irina Nurutdinova and Andrew Fitzgibbon. Towards pointless structure from motion: 3D reconstruction and camera parameters from general 3D curves. In *International Conference on Computer Vision (ICCV)*, pages 2363–2371, 2015.

[119] Tae-Hyun Oh, Kyungdon Joo, Neel Joshi, Baoyuan Wang, In So Kweon, and Sing Bing Kang. Personalized cinemagraphs using semantic understanding and collaborative learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5160–5169, 2017.

[120] Makoto Okabe, Ken Anjyo, Takeo Igarashi, and Hans-Peter Seidel. Animating pictures of fluid using video examples. In *Computer Graphics Forum*, volume 28, pages 677–686. Wiley Online Library, 2009.

[121] Luc Oth, Paul Furgale, Laurent Kneip, and Roland Siegwart. Rolling shutter camera calibration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1360–1367, 2013.

[122] Onur Ozyesil and Amit Singer. Robust camera location estimation by convex programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2674–2683, 2015.

[123] Junting Pan, Chengyu Wang, Xu Jia, Jing Shao, Lu Sheng, Junjie Yan, and Xiaogang Wang. Video generation from single semantic label map. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2019.

[124] Liyuan Pan, Yuchao Dai, Miaomiao Liu, and Fatih Porikli. Depth map completion by jointly exploiting blurry color images and sparse depth maps. In *Applications of Computer Vision (WACV), 2018 IEEE Winter Conference on*, pages 1377–1386. IEEE, 2018.

[125] Jaesik Park, Hyeongwoo Kim, Yu-Wing Tai, Michael S Brown, and In So Kweon. High-quality depth map upsampling and completion for rgb-d cameras. *IEEE Transactions on Image Processing*, 23(12):5559–5572, 2014.

[126] Jeong Joon Park, Aleksander Holynski, and Steven M Seitz. Seeing the world in a bag of chips. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1417–1427, 2020.

[127] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021.

[128] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.

[129] Shmuel Peleg, Moshe Ben-Ezra, and Yael Pritch. Omnistereo: Panoramic stereo imaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):279–290, 2001.

[130] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017.

[131] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph.*, 23(3):664–672, 2004.

[132] Marc Pollefeys, Frank Verbiest, and Luc Van Gool. Surviving dominant planes in uncalibrated structure and motion recovery. In *European Conference on Computer Vision (ECCV)*, pages 837–851, 2002.

[133] Ekta Prashnani, Maneli Noorkami, Daniel Vaquero, and Pradeep Sen. A phase-based approach for animating images using video examples. In *Computer Graphics Forum*, volume 36, pages 303–311. Wiley Online Library, 2017.

[134] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[135] J.M.M. Montiel R. Mur-Artal and Juan D. Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[136] Christian Richardt, Douglas Orr, Ian Davies, Antonio Criminisi, and Neil A Dodgson. Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid. In *European conference on Computer vision*, pages 510–523. Springer, 2010.

[137] Carsten Rother. Linear multi-view reconstruction of points, lines, planes and cameras using a reference plane. In *International Conference on Computer Vision (ICCV)*, 2003.

[138] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314, 2019.

[139] Grant Schindler and Frank Dellaert. Atlanta world: An expectation maximization framework for simultaneous low-level edge grouping and camera calibration in complex man-made environments. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* IEEE, 2004.

[140] Arno Schödl, Richard Szeliski, David H Salesin, and Irfan Essa. Video textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 489–498. ACM Press/Addison-Wesley Publishing Co., 2000.

[141] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

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

[143] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 145–150, 2014.

[144] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *null*, pages 519–528. IEEE, 2006.

[145] Jonathan Shade, Steven Gortler, L. He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM Press/Addison-Wesley Publishing Co., 1998.

[146] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a generative model from a single natural image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4570–4580, 2019.

[147] Qi Shan, Brian Curless, Yasutaka Furukawa, Carlos Hernández, and Steven M. Seitz. Occluding contours for multi-view stereo. *Conference on Computer Vision and Pattern Recognition*, pages 4002–4009, 2014.

[148] Qi Shan, Brian Curless, Yasutaka Furukawa, Carlos Hernandez, and Steven M Seitz. Photo uncrop. In *European Conference on Computer Vision*, pages 16–31. Springer, 2014.

[149] Armon Shariati, Bernd Pfrommer, and Camillo J Taylor. Simultaneous localization and layout model selection in manhattan worlds. *IEEE Robotics and Automation Letters*, 4(2):950–957, 2019.

[150] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[151] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. In *Advances in Neural Information Processing Systems*, pages 7137–7147, 2019.

[152] Andreas Simon, Randall C Smith, and Richard R Pawlicki. Omnistereo for panoramic virtual environment display systems. In *IEEE Virtual Reality 2004*, pages 67–279. IEEE, 2004.

[153] Sudipta N. Sinha, Drew Steedly, and Richard Szeliski. A multi-stage linear approach to structure from motion. In *ECCV 2010 Workshop on Reconstruction and Modeling of Large-Scale 3D Virtual Environments*, 2010.

[154] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2019.

[155] Noah Snavely. Bundler: Structure from motion (SfM) for unordered image collections. *Code available at http://phototour.cs.washington.edu/bundler/*, 2010.

[156] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):835–846, 2006.

[157] D. Steedly, I. Essa, and F. Dellaert. Spectral partitioning for structure from motion. In *International Conference on Computer Vision (ICCV)*, 2003.

[158] Henrik Stewenius, Christopher Engels, and David Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[159] Julian Straub, Guy Rosman, Oren Freifeld, John J. Leonard, and John W. Fisher, III. A mixture of Manhattan frames: Beyond the Manhattan world. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[160] Jan Stühmer, Stefan Gumhold, and Daniel Cremers. Real-time dense geometry from a handheld camera. *Proceedings of the 32Nd DAGM Conference on Pattern Recognition*, pages 11–20, 2010.

[161] Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[162] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018.

[163] Chris Sweeney, Aleksander Holynski, Brian Curless, and Steve M Seitz. Structure from motion for panorama-style videos. *arXiv preprint arXiv:1906.03539*, 2019.

[164] Christopher Sweeney, Tobias Hollerer, and Matthew Turk. Theia: A fast and scalable structure-from-motion library. In *ACM International Conference on Multimedia*, pages 693–696, 2015.

[165] Richard Szeliski. Locally adapted hierarchical basis preconditioning. *ACM Trans. Graph.*, 25(3):1135–1143, 2006.

[166] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, New York, 2010.

[167] Richard Szeliski et al. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2007.

[168] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 251–258, 1997.

[169] Richard Szeliski and Philip HS Torr. Geometrically constrained structure from motion: Points on planes. In *European Workshop on 3D Structure from Multiple Images of Large-Scale Environments*, pages 171–186. Springer, 1998.

[170] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020.

[171] James Tompkin, Fabrizio Pece, Kartic Subr, and Jan Kautz. Towards moment imagery: Automatic cinemagraphs. In *2011 Conference for Visual Media Production*, pages 87–93. IEEE, 2011.

[172] Philip HS Torr and Andrew Zisserman. Robust parameterization and computation of the trifocal tensor. *Image and Vision Computing*, 15(8):591–605, 1997.

[173] 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, 1999.

[174] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 551–560, 2020.

[175] Tinne Tuytelaars, Krystian Mikolajczyk, et al. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.

[176] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

[177] Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2443–2451, 2015.

[178] Sheng-Yu Wang, Oliver Wang, Andrew Owens, Richard Zhang, and Alexei A Efros. Detecting photoshopped faces by scripting photoshop. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10072–10081, 2019.

[179] Ting-Chun Wang, Ming-Yu Liu, Andrew Tao, Guilin Liu, Jan Kautz, and Bryan Catanzaro. Few-shot video-to-video synthesis. *arXiv preprint arXiv:1910.12713*, 2019.

[180] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional GANs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[181] Yifan Wang, Aleksander Holynski, Xiuming Zhang, and Xuaner Cecilia Zhang. Sunstage: Portrait reconstruction and relighting using the sun as a light stage. *arXiv preprint arXiv:2204.03648*, 2022.

[182] Chamara Saroj Weerasekera, Thanuja Dharmasiri, Ravi Garg, Tom Drummond, and Ian Reid. Just-in-time reconstruction: Inpainting sparse maps using single view depth predictors as priors. *arXiv preprint arXiv:1805.04239*, 2018.

[183] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SynSin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7467–7477, 2020.

[184] Kyle Wilson and Noah Snavely. Robust global translations with 1DSfM. In *European Conference on Computer Vision (ECCV)*, pages 61–75, 2014.

[185] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8534–8543, 2021.

[186] Bartlomiej Wronski, Ignacio Garcia-Dorado, Manfred Ernst, Damien Kelly, Michael Krainin, Chia-Kai Liang, Marc Levoy, and Peyman Milanfar. Handheld multi-frame super-resolution. *ACM Transactions on Graphics (TOG)*, 38(4):1–18, 2019.

[187] Changchang Wu. VisualSFM: A visual structure from motion system. *Code available at http://ccwu.me/vsfm/*, 2011.

[188] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3057–3064, 2011.

[189] Wei Xiong, Wenhan Luo, Lin Ma, Wei Liu, and Jiebo Luo. Learning to generate time-lapse videos using multi-stage dynamic generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2364–2373, 2018.

[190] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in neural information processing systems*, pages 91–99, 2016.

[191] Hang Yan, Yebin Liu, and Yasutaka Furukawa. Turning an urban scene video into a cinemagraph. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 394–402, 2017.

[192] Shichao Yang and Sebastian Scherer. Monocular object and plane slam in structured environments. *IEEE Robotics and Automation Letters*, 4(4):3145–3152, 2019.

[193] Mei-Chen Yeh and Po-Yi Li. An approach to automatic creation of cinemagraphs. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 1153–1156. ACM, 2012.

[194] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

[195] Guofeng Zhang, Jiaya Jia, Tien-Tsin Wong, and Hujun Bao. Consistent depth maps recovery from a video sequence. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(6):974–988, 2009.

[196] Jiangning Zhang, Chao Xu, Liang Liu, Mengmeng Wang, Xia Wu, Yong Liu, and Yunliang Jiang. DTVNet: Dynamic time-lapse video generation via single still image. *arXiv preprint arXiv:2008.04776*, 2020.

[197] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[198] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[199] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape-from-shading: a survey. *IEEE transactions on pattern analysis and machine intelligence*, 21(8):690–706, 1999.

[200] Yinda Zhang and Thomas Funkhouser. Deep depth completion of a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 175–185, 2018.

[201] Huizhong Zhou, Danping Zou, Ling Pei, Rendong Ying, Peilin Liu, and Wenxian Yu. Structslam: Visual slam with building structure lines. *IEEE Transactions on Vehicular Technology*, 64(4):1364–1375, 2015.