MANIPULATING SPACE AND TIME IN VISUAL MEDIA

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sean J. Liu
August 2023

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Maneesh Agrawala, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

James Landay

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Aaron Hertzmann

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.

Abstract

With the increased usage of digital technology, visual media has become a popular form of communication and is widely used for storytelling and art. Often times, authors of visual media may wish to make spatial or temporal edits in post-production. However, it can be difficult to author edits while preserving realism. One main issue is that there are many constraints involved in realism, which limit the edits that can be achieved given user-specified inputs. Moreover, often times these constraints are not explicitly defined. In this work, we introduce *task-dependent realism*, which explicitly defines realism for a target manipulation task. We focus on two manipulation tasks and identify spatial and temporal properties to relax to achieve a greater number of realistic-looking edits. This thesis contributes: 1) spatial and temporal constraints to relax and maintain for two manipulation tasks, based on perceptual properties; and 2) techniques which automatically maintain and relax these constraints as the user specifies input constraints and explores edits.

Acknowledgments

I have learned and grown so much throughout my Ph.D. journey. I'm grateful for all the people who I have encountered along the way. First, I'd like to thank my advisor, Maneesh Agrawala, for his support throughout the years. I'd also like to thank my research mentors Aaron Hertzmann and Stephen DiVerdi – I would not be where I am today without their mentorship and guidance. I'd also like to thank my committee members James Landay, Ron Fedkiw, and Sean Follmer for their support and feedback. Lastly, I'd like to thank Michael Bernstein for providing me with invaluable teaching and learning experiences during my final year as a Ph.D. student.

I am fortunate to have an amazing cohort of researchers and friends at Stanford. They made my grad school life so much more fun and exciting. In no particular order: Kevin Li, Ante Qu, Zhenglin Geng, Jane E, Ed Quigley, Winnie Lin, Minjae Lee, Mike Bao, Mackenzie Leake, Dan Fu, Jingyi Li, Mitchell Gordon, Dae Hyun Kim, Jane Wu, Jacob Ritchie, Jeongyeon Kim, Jean-Peïc Chou, Sharon Zhang, Terrell Ibanez, Jiaju Ma, and many other wonderful friends I've met at Stanford.

I'm also grateful for the wonderful friends I have outside of Stanford. Thank you for reminding me about life beyond research and for all the adventures: Katie Park, Kristi Park, Shena Fortozo, Cassie Chin, Stephanie Chin, Shirley Zhou, Justin Ying, Dai Yang, and many others.

Finally, I'd like to thank my family and especially my mother, Ruay Ho, for her emotional support throughout my Ph.D. and for her wisdom about life.

This marks the end of a chapter as well as the beginning of a new chapter, and I'm glad to say that my journey has made me more resilient, confident, and brave for the road ahead. To whomever is reading this: may you always be happy, adventurous, and courageous in all of your journeys.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rise of digital technology, visual media has become an important and popular form of communication and is widely used for storytelling and art. Capturing and displaying visual media has become more accessible than ever before. Often times, authors may wish to edit captured media in post-production to achieve various artistic or storytelling goals. This is evident by the wide range of editing tools available today, such as Adobe Photoshop [2] and Adobe Premiere Pro [3].

One common objective when editing visual media is to preserve *realism*. For instance, when photoshopping a person into an photo, the author may want the resulting composite image to look "realistic." But what is realism? Despite the long history of computer graphics in pursuing realism, there is no consensus on an explicit definition. Rather, often times, we resort to an *implicit*, *negative* definition of realism: a *lack* of distortions or clues that the media was not captured "as is." However, this implicit, negative definition of realism places the burden of maintaining realism onto the user. As users explores edits, in addition to specifying input parameters for the editing task, they also have to check if those parameters have produced a realistic-looking output. As a result, users often resort to a trial-and-error approach for maintaining realism while also trying to achieve their authoring goals, which adds friction to the authoring process.

To alleviate this burden from users, one approach is to encode realism as explicit constraints in the editing tool and automatically enforce them as the user explores edits. However, in general, this is challenging to do because there are many constraints involved in realism, and it is non-trivial to define them all explicitly. Even if one were able to list all of them out, the large number of constraints of realism would significantly limit the edits that can be achieved given user-specified inputs.

Rather than aiming for a general definition of realism, if we focus on a specific task, we can more easily identify a set of explicit constraints to relax and maintain for that task. By identifying explicit constraints to *relax*, we can increase the space of realistic-looking outputs which the user can explore. By identifying explicit constraints to *maintain*, we can design algorithms to automatically

preserve realism as the user explores edits.

This thesis introduces *task-dependent realism*, which explicitly defines realism for a target manipulation task. More specifically, task-dependent realism consists of:

- a set of explicit constraints to relax, R;
- a set of explicit constraints to maintain, M;
- a set of user-specified input parameters, I

for a target manipulation task, such that, when R and M are applied, manipulating I would generate realistic-looking outputs. For the given task and I, R represents properties that, when relaxed, would still produce realistic-looking results, and M represents important perceptual properties to maintain.

This work focuses on two target manipulation tasks of visual media – one that manipulates *space*, and one that manipulates *time* – and for each task identifies properties to relax and maintain. This thesis contributes: 1) task-dependent realism constraints for two target manipulation tasks, and 2) techniques that apply these task-dependent realism constraints as the user specifies input constraints and explores edits.

1.1 Tasks Overview

The two manipulation tasks in this thesis involve spatial and temporal editing in visual media while preserving realism. The first task manipulates space in photographs by changing object size, foreshortening, and position. The second task manipulates time in 360° video by seamlessly adjusting video time based on the viewer's head motion to achieve storytelling goals. Here, we give a brief overview of the tasks and associated challenges.

1.1.1 Editing Photographic Composition

The first task is to adjust relative object sizes and positions in photographs. Many of us can probably relate to this problem: when taking a photo of a distant object, such as the moon in the sky, the object appears smaller in the photograph than it does in real life. While a naïve solution is to zoom in and crop, this approach inevitably cuts out scene elements in the periphery, which may be undesirable. In this task, our goal is to adjust object sizes and positions while preserving realism and maintaining the visibility of user-selected scene elements.

To achieve this, we identify a set of task-dependent realism constraints and design a new image manipulation tool, *ZoomShop*, to apply these constraints as the user makes edits. For this task, we show that the important properties to maintain (M) are depth structures (including depth continuity and ordering). The constraints which we can relax (R) are light ray properties; specifically, we show that even if light rays are not straight, we can still produce realistic-looking results for this task. By

relaxing these constraints, we are able to achieve changes at larger scales than was possible before. The input parameters (I) for this task are the selected pixels and amount of change (e.g., scale or offset), both of which are designated by the user.

By applying the identified task-dependent realism constraints, ZoomShop increases the space of candidate outputs and generates realistic-looking results as the user makes edits. We discuss these constraints and techniques in Chapter 3.

1.1.2 Adaptive Playback of 360° Video

The second task is to adapt the playback time of 360° videos based on where the viewer looks in order to achieve storytelling goals. In 360° video, viewers can look anywhere in the 360° scene at any time. However, this freedom also poses storytelling challenges for filmmakers. In particular, viewers of 360° video may miss important story events if they were looking in the wrong direction. We introduce *gated clips* to help filmmakers achieve storytelling goals by guaranteeing that viewers see important story events in 360° video. Gated clips only play past a certain timestamp (*gate time*) if the viewer satisfies a viewing condition (*gate condition*). If the viewer is looking in the wrong direction, gated clips wait for the viewer by adaptively extending the length of the video until the viewer looks in the right direction.

In this task, our goal is to generate gated clips by finding seamless transitions in the 360° video to jump back in time. During playback, if the viewing condition is not met, gated clips play the selected transitions to jump back in time and extend the playback length of the video until the viewing condition is met (e.g., viewer looks in the right direction). For this task, we identify a set of task-dependent realism constraints and design techniques to apply these constraints as the user specifies gate parameters. In particular, we show that the important properties to maintain (M) are within-view spatiotemporal coherency, i.e., the visual content within the viewer's field of view looks coherent in space and time. The properties which we can relax (R) are linear time and out-of-view spatiotemporal coherency. In other words, the visual content outside of the viewer's field of view does not need to be coherent, and the timeline of the gated clip does not need to adhere the originally captured linear timeline. The user-specified input parameters (I) are the gate time, a region of interest (ROI), and a condition.

By applying the identified task-dependent realism constraints, gated clips increase the space of candidate outputs and generates realistic-looking results as the user explores edits. We discuss these constraints and techniques in Chapter 4.

1.2 Broader Impact

While this thesis focuses on two manipulation tasks and the methods of applying task-dependent realism for these specific tasks, it showcases the advantage of task-dependent realism at a broader

level. By explicitly identifying and applying constraints to relax and maintain for a given task, we can design editing tools to increase the space of realistic-looking solutions and automatically generate realistic results as the user explores edits. We speculate that task-dependent realism offers a new way of editing visual media – by decoupling the properties of realism from user-specified input parameters, task-dependent realism alleviates the burden of maintaining realism from the user and allows them to focus on authoring goals when exploring edits.

1.3 Statement of Published Papers and Multiple Authorship

This thesis is based on the publications: *ZoomShop: Depth-Aware Editing of Photographic Composition* [88] and *View-Dependent Video Textures for 360° Video* [87]. I am the lead author on these publications, but this research could not have been completed without my advisor, Maneesh Agrawala, and research mentors, Stephen DiVerdi and Aaron Hertzmann.

1.4 Roadmap

I will begin with a background overview of realism in computer graphics, visual media editing, as well as other related work (Chapter 2). Then, I will dive into the details of the two projects: *ZoomShop* (Chapter 3) and *Gated Clips* (Chapter 4) and conclude with some directions of future work (Chapter 5).

Chapter 2

Related Work

In this chapter, I will first discuss broader work related to task-dependent realism and then discuss work specifically related to the two manipulation tasks explored in this thesis.

2.1 Realism in Computer Graphics

Realism has long been pursued throughout the history of computer graphics. While there is no consensus on a general definition of realism, there have been several proposals [45, 22, 77]. These proposals generally fall under two categories which aim to model different aspects of reality: 1) the physical properties of the real world (e.g., does an image accurately follow light paths), and 2) the perceptual properties (e.g., does an image evoke the same visual response from viewers as the real scene). Ferwerda et al. [45] called the former *physical* realism and the latter *photo*-realism. Each view of realism sets different goals and gave rise to different groups of computer graphics techniques which optimize for these goals.

Examples of techniques that follow the first definition of realism and model the world based on physical properties include **physics-based simulation** [21]. Many works in this domain have looked into the simulation of various phenomenon, such as fire [101], water [73, 42], smoke [43], and trees [111]. The underlying assumption is that the physical properties of real world *are* the important properties of realism: by accurately modeling the physical properties, we get closer to visually replicating reality. However, the accuracy of physics-based modeling may not directly correlate with perceived realism. One example of this phenomenon is the uncanny valley, where a more physically accurate modeling of the human face leads to eerier effects [123, 62].

Another group of techniques optimizes for the perceptual properties of the real world [115, 32]. Many **perception-based methods** aim to quantify the attributes of an image that make it look perceptually real [112, 89, 161] and use it for improving realism [150]. Human visual perception filters the information it receives and creates some flexibility for rendering synthetic imagery [115]. Many

computer graphics techniques have built upon this insight, such as selective rendering [10, 37, 138], attention redirection [132], and efficient image and video processing [149]. All these methods leverage features of human visual perception to achieve realism.

Other techniques optimize for both physical and perceptual properties of the real world. **Data-driven methods**, for example, measure realism based on how closely an image matches the distribution of existing data, which may capture both physical and perceptual properties. Today, many machine learning techniques are able to learn general distributions from existing data [41] and use them to generate photorealistic results [53]. While powerful, these techniques learn an implicit, black-box representation of realism (e.g., via discriminators) rather than explicit constraints of realism.

In this thesis, we define and explore task-dependent realism, which identifies properties and constraints to optimize for a specific manipulation task, rather than adhering to a general definition of realism. The identified constraints of task-dependent realism are based on the perceptual properties of realism. Unlike many other perception-based methods, however, our work focuses on specific tasks and on identifying and applying realism constraints for those tasks, rather than quantifying visual perception or coming up with general heuristics that exploit visual perception. The task-specific nature allows us to aggressively relax physical properties for a specific task that would still produce realistic outputs. While the set of relaxed properties for one task may not generalize to other tasks, it significantly broadens the range of realistic edits the user may produce and enables powerful editing techniques. In addition, task-dependent realism defines realism in terms of explicit constraints of the task rather than general black-box, implicit representations such as those from data-driven methods.

2.2 Exploring Realistic Edits

Many works have proposed editing techniques for visual media while preserving realism. One large body of work is machine learning models for image editing. Many of these models are powerful at generating realistic-looking images [53] by learning an implicit manifold of real images, but they offer varying levels of user control. Earlier models have less fine-grained controls and appealed to more automatic ways of image editing, such as domain translation [68, 163, 164, 36], image composites [51], and other enhancements [116]. Many techniques train specific models and components for a given task, such as image blending and texture transfer [105, 148, 159]. Nowadays, there is a lot of research that provide more control over the model outputs, such as through latent vector editing [162, 60, 109, 140, 11], direct parameter editing [35] and semantic map labels [145]. More fine-grained controls include sketch-based [65, 94], text-based [165, 50, 98], exemplar-based [151], and video-driven [74] approaches.

While these machine learning models are powerful, they learn an implicit, general distribution

of realistic-looking images rather than explicit constraints of realism for those tasks. Since they learn statistical distributions, the outcomes are not guaranteed to be realistic. When the generated outputs fall short of realism, it is challenging to understand why and make necessary adjustments due to the statistical, implicit definition of realism. On the other hand, our approach explicitly defines realism constraints for a specific task, including constraints to relax in order to increase the space of realistic edits the user may explore.

Another group of techniques uses an optimization-based approach for visual media editing [7, 13, 31, 30]. These optimizations typically implicitly encode a definition of realism through their constraints or objectives. For example, seam carving [13] offers a framework for removing or adding seams in an image based on user-defined operators, such as low changes in gradient. Similarly, digital photomontage [7] combines a stack of images based on user-defined objectives, such as color similarity. In these cases, realism is defined implicitly via low-level heuristics (i.e., high-frequency spatial regions and color consistency) selected by the user. Although the frameworks can generalize to several types of images and tasks, they also place the burden of maintaining realism on users. Other methods focus on more specific tasks and build in more constraints for maintaining realism. For example, some image warping methods [31, 30] include smoothness and conformality constraints in their optimizations. Our work is similar in that we focus on specific tasks and identify explicit constraints to encode in optimizations. However, in addition to identifying constraints for maintaining realism, the work in this thesis places additional emphasis on identifying constraints to relax in order to increase the range of feasible edits for that task.

2.3 Related Work: ZoomShop

In the first project of this thesis, we introduce an image manipulation tool, *ZoomShop*, for adjusting relative object sizes and positions. ZoomShop uses knowledge of the image's 3D geometry to provide the specific controls needed to edit photographic composition while maintaining important image structures. Here, we discuss related work in image retargeting and editing.

Many image retargeting techniques do not consider depth. Cropping [134] can make an object appear larger in the image but cuts out peripheral content. Seam carving [12, 13] removes pixels as well but focuses on non-salient ones along an image dimension. Shift maps [110], cut-and-paste [122], and patch-based image editing methods [129, 20, 124] support arbitrary reshuffling and allow new pixels to overlay existing pixels. While these image retargeting methods are powerful, they lack scene depth understanding and thus may not preserve occlusion relationships, depth structure, or provide direct control over adjusting perspective. One contribution of ZoomShop is to use depth information for warping, as depth information is becoming more readily available, either from consumer mobile phone cameras or estimated via state-of-the-art computer vision methods.

Several previous methods specifically aim to adjust scene perspective. Carroll et al. [31, 30]

introduced warping-based methods for manipulating the local perspective in an image, creating non-linear warps of a fixed camera projection. Fried et al. [49] show how to modify foreshortening in portraits by fitting a virtual perspective camera and then warping the image. Other transformation techniques [166, 128, 31] minimize distortions in wide-angle images by combining different perspectives in different regions of the image. Other works aim to create realistic-looking panoramas through perspective deformation [6, 156, 125]. These methods are all constrained to produce one-to-one mappings between the input and output images, which means they do not support changes in occlusion relationships between objects. This fundamentally limits their ability to deform scenes involving both foreground and background elements, a common aspect of photographs that ZoomShop addresses.

Our method is also related to work that reprojects a single image using depth estimation and conventional projection models [127, 76]. Niklaus et al. [103] recreate "Ken Burns" effects by translating a linear perspective camera through a photograph's 3D scene. These methods all utilize conventional linear perspective camera models, which limits their results to only geometrically accurate reprojections. ZoomShop on the other hand uses a non-linear camera model and warp, enabling a wider class of image transforms that better supports photographers compositional goals, such as enlarging a distant object while keeping foreground elements fixed.

Most closely related to ZoomShop is the work of Badki et al. [14], which combines multiple photographs to reproject the scene with a piecewise linear camera. We extend their camera model by including linear, curved, and discontinuous segments in our camera model, which support different types of edits and scenes. ZoomShop also uses a simpler workflow, relying on a single image with a measured or estimated depth map, which enables editing a broader range of images, such as dynamic scenes or historical events.

Other techniques have been proposed for combining perspectives from multiple cameras to create a single non-linear artistic projection [9, 48, 130, 39, 155, 38, 153], or by non-linear ray tracing [25, 24, 90, 154]. General purpose non-linear camera models are very powerful and can produce a wide range of image appearances. Our camera model is a subset of these general models, specifically designed for ease of editing and maintaining scene spatial relationships. We also use a depth-aware rubber sheet warp to support scene element translation. Our results could be created by a more general non-linear camera model alone; our contribution is the alignment between ZoomShop's user controls and its underlying implementation.

Finally, our work is motivated in part by an intriguing observation from the perception literature: in some cases linear perspective photographs may not capture subjective visual experience as well as nonlinear images, particularly those that inflate the size of objects in or near the fovea [18, 27, 114, 75]. Allowing users to resize objects in images could be useful for better conveying scenes and describing visual experiences [93].

2.4 Related Work: Gated Clips

In the second project of this thesis, we discuss a new filmmaking technique, *gated clips*, designed to ensure that a viewer sees key elements of the narrative in a 360° video. Using this technique, the filmmaker can author a *gate* which ensures that playback may only proceed past the gate if a viewing condition is met. To enforce gates, our technique seamlessly loops 360° video playback until the gate condition is met. Here, we discuss three main areas of related work.

2.4.1 Interactive and looping video

Forms of interactive (or dynamic) video, where the video playback changes depending on viewer actions, have been explored for several decades. The first methods, including Movie-maps [84] and QuickTimeVR [33], allowed navigation in real and virtual environments, whereas some arcade video games, such as Space Ace [5] and Dragon's Lair [4], used interactive video for branching narratives.

Our method uses video textures to create video that loops seamlessly until certain conditions are met. Video textures were introduced by Schödl et al. [120], who demonstrated finding seamless non-repeating paths through short clips to create the experience of endlessly playing videos, as well as to drive video in different ways. This technique has been extended to loop video panoramas captured by a panning camera [8], motion capture videos of humans [46], videos orbiting around a moving object [78], and responsively looping video based on user interactions [66].

In order to increase the number of possibilities for seamless transition in a video texture, some methods have segmented the video into independently-moving regions [120, 59], and then created separate video textures for each such region. In contrast, we introduce *view-dependent arcs*, specifically for applying video textures to 360° video. Our approach is complementary to segmentation; our method produces video textures by exploiting viewers' limited FOV in 360° video, whereas segmentation loops the entire scene by looping and compositing different moving parts of the scene.

Motion segmentation methods are also used in the construction of cinemagraphs [15, 16, 71, 81, 80], photograph-like images with some moving elements. These methods create pixelwise textures that are not appropriate for our structured scenes. Our view-dependent arcs are also akin to Chenney and Forsyth's [34] view-dependent approach to accelerating physical simulation, by eliding computation for scene elements not in view.

2.4.2 Gaze guidance

While 360° video provides a new dimension of viewer interactivity and agency, this freedom is accompanied by users struggling with not knowing where to look [142, 107]. Methods for guiding or forcing the viewer to look at a particular region of interest (ROI) are called *gaze guidance*. Our approach is complementary to gaze guidance, and we expect our method to be used together with

gaze guidance techniques, such as audio cues to attract the viewer's attention. For a survey and taxonomy of gaze guidance techniques, see Nielsen et al. [102].

Subtle gaze guidance techniques provide visual and audio cues that attempt to guide the viewer's gaze without breaking immersion. This is particularly important for narrative content where a filmmaker wishes to keep the viewer fully absorbed by the story. Conventional filmmakers aggressively use subtle cues to guide the viewer's attention, and these effects are also being explored by 360° filmmakers and researchers [118, 126]. Several techniques apply subtle gaze guidance to normal field of view (FOV) video, i.e., on a desktop computer monitor. These methods include gradually blurring non-ROI parts of the image [58], applying shallow depth of field [133], modulating the scene's visual saliency [143], and presenting a small flickering distractor in the viewer's peripheral vision in the direction of the ROI [17]. For in-headset VR, subtle techniques in the literature include inducing optical flow in peripheral vision [26] and flickering elements in peripheral vision [55]. Grogorick et al. [56] perform an evaluation comparing different subtle guidance techniques. Each of these techniques finds some success in guiding viewer gaze, but none guarantee that viewers will see critical moments.

Several methods actively control the viewer's gaze direction to focus on a ROI; depending on how this is implemented, it can substantially break immersion. This includes directly rotating the scene [82] or rotating the user in a motorized swivel chair [57]. Attempts to hide this rotation include applying very slow rotations [131], or applying gains to the user's own rotation [158]. Reorienting the scene during cuts [108] maintains immersion but can only be applied at cuts. A picture-in-picture visualization may also be shown of the ROI so that viewers always see important content [83].

2.4.3 View-dependent 360° video and animation

Our work is inspired by several short 360° video and animations that use different kinds of view-dependent playback. Several animated shorts use forms of gating to pause the action (without pausing the motion or audio), either waiting for the viewer to look at something, or to look away from something, including "Buggy Night," "Piggy," and "The Simpsons: Planet of the Couches," from Google Spotlight Stories[1] [54]. In "Buggy Night" and "Piggy," the viewer is essentially part of the story, either scaring the flies by looking at them in "Buggy Night," or catching Piggy stealing a cake. The ending of the "Batman: Arkham VR" [117] video game uses an offscreen action effect to simulate the player's hallucinations. Disney's "Cycles" [144] animation fades out the lighting and action whenever the viewer looks away. Each of these examples is animated, and, presumably, carefully hand-authored to run in real-time rendering engines; whereas our method can work with live action video with comparatively lightweight authoring effort.

We are aware of only one example using live action video: "Wild: The Experience" [44] from

[1]These shorts are viewable in the Google Spotlight Stories app for iOS and Android. On YouTube, they do not have view-dependent playback. "Piggy" also has a separate app in the Steam Store.

Felix & Paul Studios. In the original short, a character may appear and disappear after certain times, depending on the viewer's head movements. The video plays for a fixed duration regardless; there is no gating, and thus no guarantee that the viewer will see significant events.

Chapter 3

ZoomShop

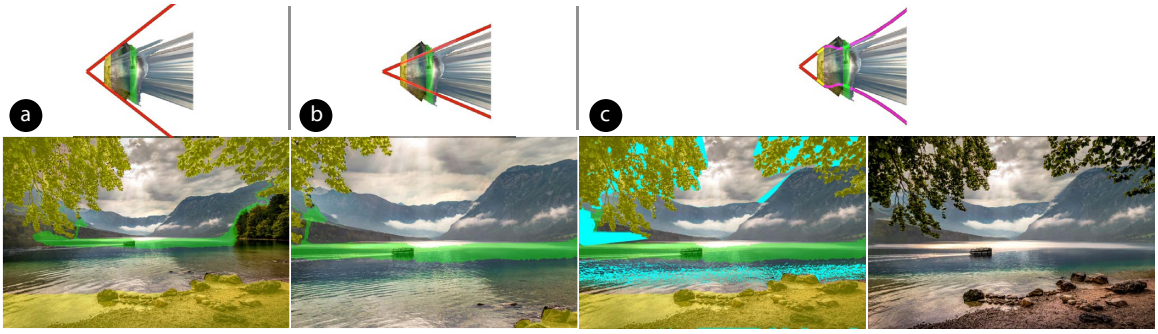


Figure 3.1: Using ZoomShop to edit a photograph [139]. (a) An image of mountains (background) framed by trees (foreground). The user's goal is to make the boat bigger while keeping the framing of foreground trees. (b) Zooming in and cropping scales up the boat, but cuts out most of the trees and shore. (c) Left: With ZoomShop users can select depth ranges (yellow and green) and independently adjust each region while maintaining scene structure. Disoccluded and stretched pixels are shown in cyan. Right: Cyan pixels from image are manually inpainted using Photoshop's Content-Aware Fill. We show the top-down view volume and boundary curve of each camera above the corresponding images in a, b, and c.

In this chapter, we explore the task of adjusting relative object size, foreshortening and positions in photographs. Consider Figure 3.1a, a photograph of a boat on a lake framed by trees. The photographer may wish to make the boat appear larger, which can be achieved by zooming, but the trees and the shore go out of frame (Fig. 3.1b). A longer lens from further back might satisfy both goals, but may be impossible due to physical constraints. With varying degrees of manual effort, today's digital tools allow adjusting the sizes of scene objects in a plausible, if not geometrically accurate, way [93]. There are many methods that can arbitrarily adjust object sizes [13, 31, 128, 20, 124], but these methods can fail to preserve important spatial relationships, which is a major component of realism.

In this work, we identify a set of task-dependent realism constraints and design a new image manipulation tool, *ZoomShop*, to apply these constraints as the user makes edits. By applying task-dependent realism, ZoomShop increases the space of realistic-looking edits and automatically preserves important realism properties while the user explores edits.

3.1 Editing Photographic Composition

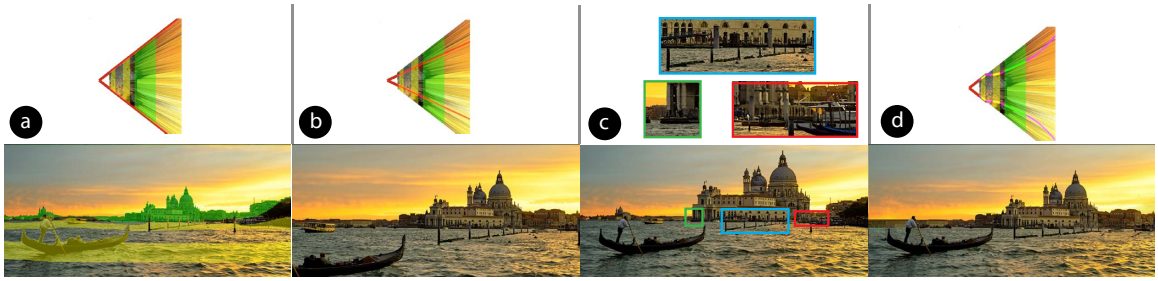


Figure 3.2: Venice [137]. Goal: scale up distant building while keeping the boat in the same place. (a) Original photo. (b) Zooming in and cropping. (c) Cutting out the building, scaling it, and pasting it back in may not preserve depth structure due to lack of scene depth understanding. Here, the building occludes side buildings and boats, and the poles in front are cut in half (marked with boxes). (d) Our result with inpainting (manual guidance).

We first understand a hypothetical photographer's editing goals with the example in Figure 3.2. Upon taking the photo, the photographer realizes the distant building, the Santa Maria della Salute ("Salute"), appears smaller in the image than they would like and wishes to make it larger.

Goal 1. Make a target object appear larger or closer.

Zooming in and cropping scales the Salute but cuts out the foreground boat (Fig. 3.2b). As composition is important to balance the image, the photographer wishes to preserve it as in the original photo.

Goal 2. Maintain scene element visibility.

Cutting out the Salute, resizing it, and pasting it back in (Fig. 3.2c) create artifacts where the docking poles in front are cut in half and nearby elements are occluded (depth ordering), and care must be taken to ensure the resized Salute contacts the unedited ground (depth continuity).

Goal 3. Maintain scene element spatial relationships.

3.1.1 Task-Dependent Realism

In this task, the user-input parameters (I) are a set of objects or regions and their desired amount of change in scale or offset (Goals 1 & 2). As shown in the above example, spatial relationships are important properties to maintain (M). Specifically, for this task, depth structures are critical to

perceived realism, which include depth continuity and ordering (Goal 3). To automatically maintain constraints M as users manipulate I, we use scene depth information. However, if we strictly adhere to linear perspective, the changes in scale and foreshortening of objects may be limited due to the scene structure. For example, in Figure 3.3a, if we fix the foreground ridge and enlarge the lighthouse while maintaining linear perspective on the scene, we cannot enlarge the lighthouse beyond the width of the cliff that the lighthouse rests on. To enlarge the lighthouse even more while maintaining realism, we need to adjust the size of the cliff itself, as well as the road leading up to the cliff because they are connected in depth (Figure 3.3c).

In this work, our insight is that we can relax light ray properties (R) to smoothly vary changes in scale across depth; even if light rays are not straight, we can still produce realistic-looking results for this task. By relaxing these constraints R, we are able to achieve changes at larger scales than was possible before and parameterize the solution space to automatically maintain M as the user manipulates I.

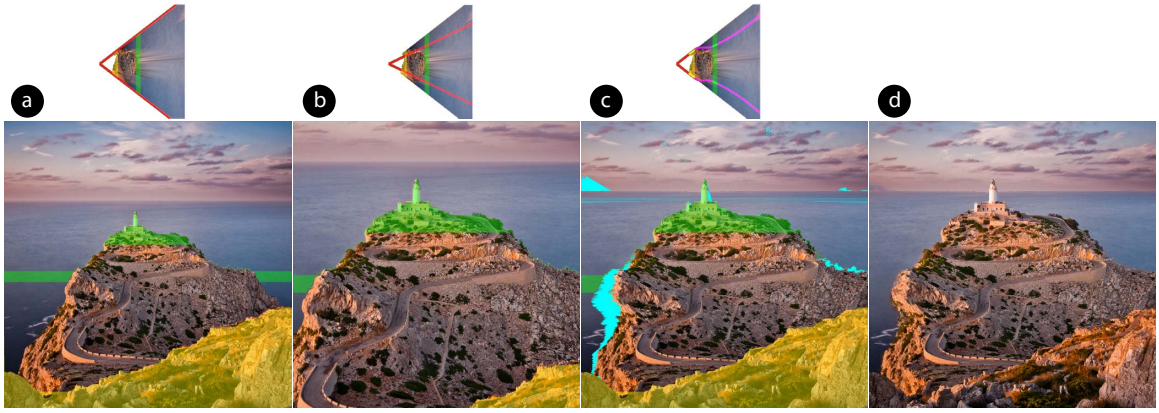


Figure 3.3: Lighthouse [97]. Goal: enlarge lighthouse while keeping compositional element of the foreground ridge. (a) Original photo. (b) Zooming in and cropping. (c) Our result. (d) Our result with inpainting (automatic).

3.1.2 Overview

We present ZoomShop, a digital image editing tool that uses knowledge of the image's 3D geometry to provide the specific controls needed to edit photographic composition while maintaining important image structures (Fig. 3.1c). With ZoomShop, users can select image regions based on depth, adjust relative scaling of each region, adjust foreshortening within each region, and translate objects horizontally within regions. These controls allow users to make high-level adjustments to the 3D structure of the image: they can enlarge distant objects, maintain foreground framing, adjust the foreshortening of an object to emphasize its 3D shape, and adjust the relative sizes and positions of objects at the same depth.

Our key technical contribution is to support user editing goals with a combination of a non-linear camera model and a depth-aware rubber sheet warp. Our novel camera model consists of a depth-varying scale function that is defined by a piecewise linear, smooth, and/or discontinuous curve. Given an input image and its depth map (either measured or estimated), ZoomShop maps the user's edits to our non-linear camera model and then reprojects the scene. Then the user can select regions and translate them horizontally. These translations define the objective function of our depth-aware rubber sheet warp, which is used to warp the image for the final result. The ZoomShop interface enables the user to select where and how deviations from linear perspective occur to support their size, position, and foreshortening goals, providing complete control over the final image appearance to the user.

We demonstrate ZoomShop's capabilities on examples, adjusting scale and foreshortening while maintaining framing, and show how it differs from other linear and non-linear imaging models. We use off-the-shelf computer vision algorithms to infer scene depth [113] and fill holes [2, 20]; we expect even better methods to be available for these subtasks in the near future, given the fast pace of progress in these fields. While we focus on outdoor scenes, ZoomShop is potentially useful for any scene exhibiting a large range of depths. We believe ZoomShop provides a useful new way to approach photography.

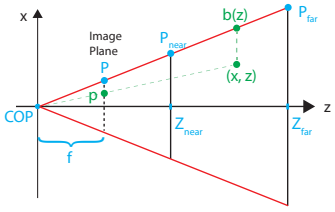
3.2 Method

To achieve the goals in Chapter 3.1, we split the problem into two complementary sub-problems: editing objects at different depths, and editing objects at the same depth. To edit objects at different depths, we reproject the image with a non-linear camera model that enables depth-dependent scaling. The user can resize an object at a target depth (Goal 1), maintain the scale (and thus visibility) of objects at other depths (Goal 2), and ensure smooth transitions between target regions to maintain spatial relationships (Goal 3). Since the scale is per-depth, objects at the same depth are all scaled the same, which may push some objects out of view. To support maintaining visibility in this case, we apply a rubber sheet warp to the image to translate peripheral objects back into view (Goal 2). The warp is also depth aware, to ensure it maintains depth continuity (Goal 3).

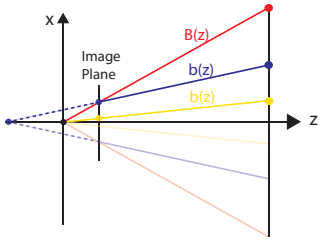
3.2.1 Editing Objects at Different Depths

To edit objects at different depths, we present a novel non-linear camera model defined by a *boundary curve* $b(z)$ that supports depth-dependent scaling.

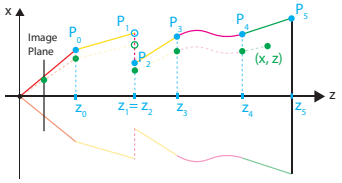
Figure 3.4a shows a top-down view of a camera frustum, with the camera at the origin. Let f be the camera focal length, (P_w, P_h) the half-width and half-height of the image plane, and Z_{near} and Z_{far} the camera near and far planes. Linear perspective maps 3D scene points $p = (x, y, z)$ to



(a) Linear Perspective. P is the half-width of the image plane in world space. Z_{near} and Z_{far} are the z-values of the near and far plane, and f is the focal length of the camera.



(b) The original boundary $B(z)$ is in red. Scaling is equivalent to zooming in and cropping (yellow). Decreasing the foreshortening of some depth range is equivalent to using a longer focal length and moving the camera back (blue).



(c) $b(z)$ with curved, linear, continuous, and discontinuous segments. To support discontinuous $b(z)$ curves, our tool allows $z_i = z_{i+1}$, $P_i \neq P_{i+1}$. In such cases, our tool sets $b(z_i = z_{i+1}) = P_{i+1}$.

Figure 3.4: Our non-linear camera model is defined by its boundary curve $b(z)$.

image coordinates $u, v \in [-1, 1]$ by

$$u = \frac{f}{P_w z} x, \; v = \frac{f}{P_h z} y \tag{3.1}$$

We allow $b(z)$ to be any function over $z \in [Z_{near}, Z_{far}]$. Given $b(z)$ and the image aspect ratio $\lambda = P_h/P_w$, the mapping from a scene point to image coordinates is

$$u = \frac{x}{b(z)}, v = \frac{y}{\lambda b(z)} \tag{3.2}$$

$b(z)$ defines the x-boundary of the view volume (Figure 3.4a), while the y-boundary is $\lambda b(z)$. Only points within the x- and y-boundaries are included in the output image: $Z_{near} \leq z \leq Z_{far}$, $-b(z) \leq x \leq b(z)$, and $-\lambda b(z) \leq y \leq \lambda b(z)$. For all boundary curves, scenes are rendered in reverse depth order (back to front) to maintain occlusion relationships.

Note that the view volume boundary $b(z)$ at a given depth z is inversely related to the scale of scene points at z in the image (Equation 3.2). Therefore, we can manipulate $b(z)$ to change the size and foreshortening of objects in the output image.

To resize an object at $p = (x, y, z)$ by a scale factor s so $(u', v') = (su, sv)$, we set $b(z) = B(z)/s$, where $B(z)$ is the input photograph's original camera boundary curve, $B(z) = P_w z/f$. This is a linear function, so applying $b(z)$ uniformly scales all image points by s and is equivalent to zooming and cropping (Fig. 3.4b). To resize the foreground and background separately, we can define $b(z)$ as a non-linear function of depth.

We can also use $b(z)$ to adjust scene foreshortening (change in size over depth) by changing

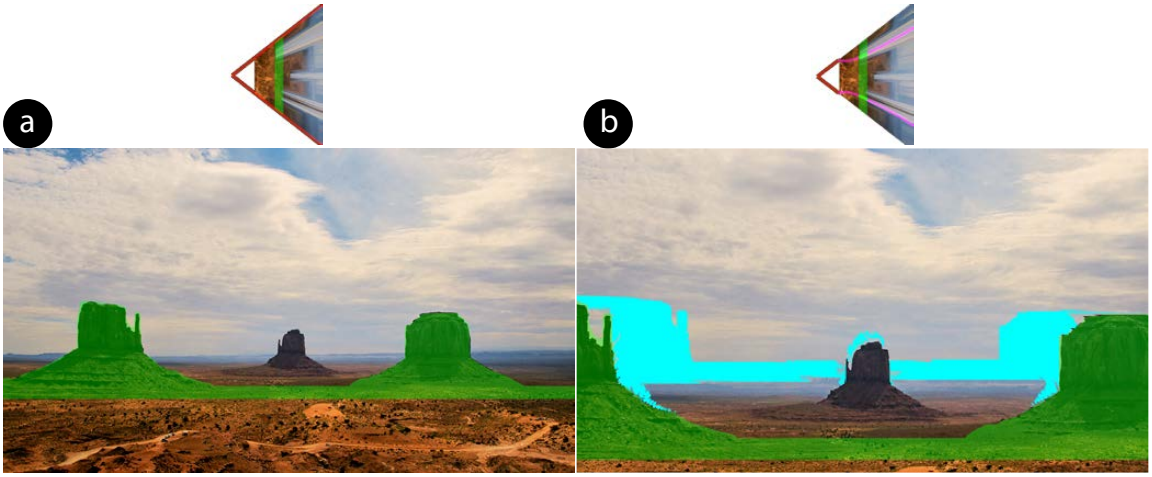


Figure 3.5: Monument Valley [136]. ZoomShop scales scene points at the same depth by the same amount. (a) Original photo. (b) Side hills (green) are scaled up but are moved partially out of view; disoccluded pixels are shown in cyan. To fix this, ZoomShop includes a depth-aware image warp to translate the side hills back into view.

$db(z)/dz$ over some range (Fig. 3.4b). For example, decreasing the slope of $b(z)$ decreases foreshortening, compressing depth and making background objects appear closer. This is equivalent to increasing the camera's focal length while pulling the camera back (as in a dolly-zoom shot). With a non-linear $b(z)$, we can achieve this result without moving the camera, and we can also independently adjust the foreshortening of different depth ranges.

Our camera model supports image formation under Equation 3.2 for $b(z)$ composed of any number of segments that are straight, curved, or discontinuous (Fig. 3.4c). Straight segments yield local linear perspective, but the image regions where different straight segments abut may have visible bending artifacts. Curved segments can be used to create smooth transitions between other segments by maintaining tangent continuity of $b(z)$, but will not preserve straight lines. Discontinuities occur when two $b(z)$ segments abut at different depths, creating a jump in scale in the output image. Discontinuities are useful when image features (e.g. texture or occlusions) allow hiding the transition. Choosing the correct set of segments depends heavily on the scene and desired results, so ZoomShop presents these options to the user.

3.2.2 Editing Objects at the Same Depth

Our camera model scales all points at the same depth uniformly. While this helps maintain spatial relationships between nearby objects, it can also move peripheral objects out of the image. For example, Figure 3.5 has two hills at the same depth on each side. Scaling them up pushes the hills out of view (i.e., no longer contained by $b(z)$). In order to maintain their new size *and* keep them

visible, ZoomShop includes an image warp optimization to translate objects back into view. Our optimization is depth-aware, so pixels that have similar depths are translated by similar amounts, while pixels at different depths can be translated differently.

Given user-defined regions $R_1, R_2, ...R_K$ and a corresponding desired translation $T^1, T^2, ...T^K$ for each one, we solve for a per-pixel translation map $t_{i,j}$ for each pixel i, j. To reduce the search space of $t_{i,j}$, we limit the search to horizontal translations only. This is a reasonable limitation because objects that are attached to the ground should remain on the ground after translation, but it is straightforward to extend to 2D translations if needed. For a pixel at (u_i, v_j), the new location is $(u_i + t_{i,j}, v_j)$.

Our goal is to smoothly propagate translations across continuous depths. Thus, we minimize the following objective function:

$$\min_t \lambda_s E_{\text{smooth}} + E_{\text{reg}} \tag{3.3}$$

$$\text{subject to } t_{i,j} = T^k,\ \forall i, j \in R_k,\ k = 1...K \tag{3.4}$$

where the input region translations are hard constraints and λ_s adjusts how quickly translations fall off. We used $\lambda_s = 10^5$ in all of our results.

The smoothness term encourages $t_{i,j}$ to vary smoothly across pixels at the same depth:

$$E_{\text{smooth}} = \frac{1}{N} \sum_{i,j} [w^v_{i,j}(t_{i,j+1} - t_{i,j})]^2 + [w^h_{i,j}(t_{i+1,j} - t_{i,j})]^2 \tag{3.5}$$

where $N = 2WH - W - H$ is the total number of pairs of neighboring pixels for image width W and height H.

The smoothness weights are reduced at depth discontinuities:

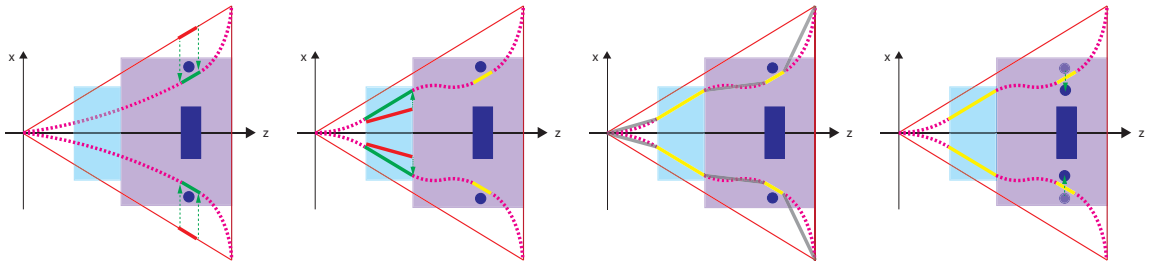
$$w^v_{i,j} = \alpha_v \boldsymbol{\sigma}(|z_{i,j+1} - z_{i,j}|) = \alpha_v \frac{1}{1 + e^{\beta(|z_{i,j+1} - z_{i,j}| - \gamma)}} \tag{3.6}$$

$$w^h_{i,j} = \alpha_h \boldsymbol{\sigma}(|z_{i+1,j} - z_{i,j}|) = \alpha_h \frac{1}{1 + e^{\beta(|z_{i+1,j} - z_{i,j}| - \gamma)}} \tag{3.7}$$

α_v and α_h adjust the relative penalty between vertically and horizontally adjacent pixels. To avoid shearing of objects resting on the ground, we penalize translation differences across vertical pixels more than horizontal pixels. We use $\alpha_v = 5, \alpha_h = 1$ for all results. The sigmoid function $\boldsymbol{\sigma}$ selects the range of depth differences where the energy term is nonzero. We empirically found that $\beta = 10^4$, $\gamma = 10^{-4}$ works well.

We additionally include a regularization term:

$$E_{\text{reg}} = \frac{1}{WH} \sum_{i,j} t^2_{i,j} \tag{3.8}$$



(a) Step 1: Enlarge the statue (blue rectangle). Selecting the depth range adds a linear segment (bold red). Increasing the scale of the selection moves the bold red segment to the bold green segment. The slope remains the same, which preserves the foreshortening of the statue.

(b) Step 2: Expand depth of the steps (light blue rectangle). Selecting the depth range adds another another linear segment (bold red). Increasing the foreshortening changes the slope of the bold red segment to the bold green segment.

(c) Step 3: Toggle between smooth and linear interpolants. The smooth mode uses a cubic interpolant to connect the linear segments (dotted magenta). The linear mode uses linear segments (solid gray).

(d) Step 4: Translate pillars (blue circles) into view. Enlarging the statue (blue rectangle) caused the two pillars to move out of frame (i.e., no longer contained by the boundary curve). The image warp optimization moves them back into view.

Figure 3.6: Example user workflow. Top-down layout of a statue (blue rectangle) flanked by pillars (blue circles) on a platform (purple region), with steps (light blue) leading up to it.

This encourages the optimization to find the smallest deformation that satisfies the user's input.

3.3 The ZoomShop Application

ZoomShop implements our non-linear camera model and depth-aware rubber sheet warp as the basis for editing an image. The full workflow requires ingesting input images, representing boundary curves, the user interface, and rendering the final result.

3.3.1 Input Geometry

We begin with an input RGB image, and generate a non-metric disparity map from MiDaS [113, 127]. We manually clean up any obvious errors using Adobe Photoshop 2021 [2], convert it to a non-metric depth map, and then use it as the depth of a per-pixel triangle mesh as our 3D scene. For each pixel disparity $d \in [0, 1]$, we compute depth $z(d) = \frac{1}{d+0.1}$, and we set the input camera's vertical field of view to $\theta_v = 55°$. While this does not produce a geometrically accurate scene, it preserves relative depth ordering and is sufficient for our needs.

3.3.2 Boundary Curve Representation

The original boundary curve of the image is $B(z) = \frac{P_w z}{f}$ and the new boundary curve after user edits is $b(z)$. We represent $b(z)$ as a series of control points (z_i, P_i), $i = 1...N$ (Fig. 3.4c). $z_{1:N}$ is a list of depths in increasing order, and $P_{1:N}$ are the boundary positions at those depths. These control points are interpolated with linear and cubic segments to form the $b(z)$ curve. Discontinuities are supported by consecutive control points sharing the same z value. Initially, there are only two control points $z_{1:2} = \{Z_{near}, Z_{far}\}$ and $P_{1:2} = \{B(Z_{near}), B(Z_{far})\}$ that define the original boundary. As the user edits the image, ZoomShop updates the control points and reprojects the image in real-time.

3.3.3 User Controls

ZoomShop presents a set of controls to edit the image appearance, which are used to construct the boundary curve and rubber sheet warp.

First, the user **selects a depth range** by clicking on a target object to edit. ZoomShop creates a new linear boundary curve segment at the target object depth with two control points z_i, z_{i+1}. All image pixels within the depth range are highlighted, and the user may refine the selection further by adjusting the start and end depths. The boundary positions for the segment are initialized to $P_i = b(z_i)$ and $P_{i+1} = b(z_{i+1})$.

After selecting an image region, the user may **adjust its scale**, which changes P_i, P_{i+1} while keeping z_i, z_{i+1} fixed. For a scale value s, $P'_i = B(z_i)/s$ and $P'_{i+1} = P'_i + (P_{i+1} - P_i)$, which preserves the slope of the linear segment.

The user may also **adjust the foreshortening** of the selected region, which controls how image size changes over depth. This is akin to changing the camera focal length, a commonly used photographic technique to compress or emphasize the depth of an image. The user adjusts the foreshortening of a region by changing the scale at the back P_{i+1} while fixing the scale at the front P_i, which changes the slope of the boundary curve segment. Fixing P_i ensures the image size does not change, allowing foreshortening and scale to be adjusted independently.

Each image region the user adjusts corresponds to a linear segment in the boundary curve. In between those segments, the user can **select the interpolants** that complete the curve. By default, ZoomShop uses a cubic interpolant to ensure the boundary curve has smooth tangents so there are no abrupt changes in scale in the output image, but this may result in straight lines in the image becoming curved. Alternately, the user can select linear interpolation, which yields a piecewise linear boundary curve that preserves straight lines within regions, but may have visible bending artifacts at region transitions.

After editing an image, the user may find that some scaled objects have moved to undesirable locations (e.g. out of the image), so they can **translate the objects** to better positions. The user selects one or more 2D rectangles and moves them horizontally to the target locations, which become the hard constraints of our depth-aware rubber sheet warp.

3.3.4 Translation Map

ZoomShop generates the per-pixel translation map from the user's edits by optimizing the constrained objective of the depth-aware rubber sheet warp. To save computation time, ZoomShop scales the image down to the same size as its depth map, computes a per-pixel translation map on the scaled-down image, and then upsamples the output translation map back to the original image size. ZoomShop uses PyTorch's L-BFGS optimizer, which can take 2–10 minutes to complete, depending on the size of the depth map.

The optimization relies on computing the change in depth for each pixel among its neighborhood, which requires depth values to be available at every pixel. However after reprojecting the image using our camera model, there are disoccluded regions that have no depth information. Therefore, while translation rectangles are visualized on both the input and scaled images, users mark and adjust the rectangles on the input image, and the translation map is also computed on the original image.

3.3.5 Image Synthesis

To synthesize the final image, ZoomShop first applies the translation map (if any) by displacing the input mesh vertices. Then ZoomShop applies a per-depth scale factor to the scene based on the new $b(z)$. Each vertex at depth z is scaled about the image center by $B(z)/b(z)$. When rendering the output image, we identify and remove disoccluded, stretched, and sheared pixels; these are shown as cyan in our results.

To fill in the removed pixels, we use Photoshop's Content-Aware Fill (CAF) [2, 20]. For some results, the fully automatic CAF works well, but for other results, it was necessary to manually specify regions for CAF to sample from when inpainting. In those cases, we show both automatically and manually inpainted results side-by-side. The final image quality depends on the inpainting, which is an active area of research [85, 152] that we expect to continue to improve rapidly.

3.3.6 Example Workflow

We give an example of a hypothetical user workflow in Figure 3.6, which shows a top-down illustrated layout of a statue (blue rectangle) flanked by pillars (blue circles) on a platform (purple region), with steps (light blue) leading up to it. Upon taking the photo, the user realizes the statue appears too small and wants to enlarge it. So they first select a depth range that contains the statue and increase its scale (Fig. 3.6a). After scaling the statue, the user realizes the foreground steps appear too short and wants to expand their depth. So they select a depth range containing the steps and increase its foreshortening (Fig. 3.6b). After modifying these two depth ranges, the user toggles between the smooth and linear interpolants (Fig. 3.6c), and finds that linear interpolation creates a noticeable bend in between the steps and the platform, while the platform does not contain visually important

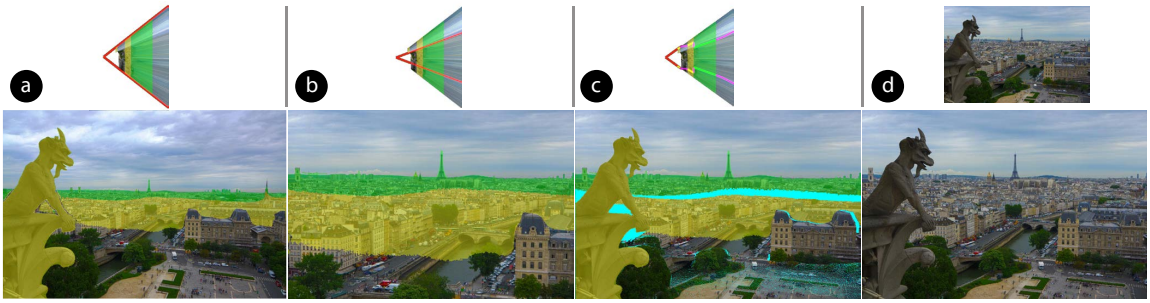


Figure 3.7: Eiffel Tower [92]. Goal: enlarge the Eiffel Tower while keeping the Gargoyle visible. (a) Original photo. (b) Zooming in and cropping. (c) Our result. (d) Our result with inpainting (top: automatic; bottom: manual guidance). We break the scene up into three depth ranges; the gargoyle, the intermediate buildings, and the Eiffel Tower. The intermediate depth range maintains linear perspective for the buildings, and we place the discontinuity between the far two depth ranges in a highly textured region for easier inpainting.

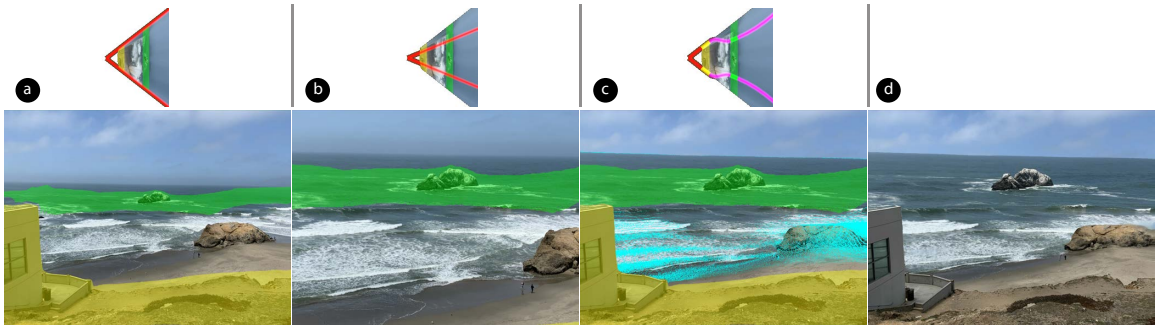


Figure 3.8: Seal Rocks. Goal: enlarge Seal Rocks (background) while keeping Cliff House (foreground, left) in view. (a) Original photo. (b) Zooming in and cropping. (c) Our result. (d) Our result with inpainting (automatic).

straight lines. Therefore, they select smooth interpolation. Finally, the user finds that the pillars next to the statue have been moved out of the image, so they select the pillars and translate them back into view (Fig. 3.6d).

3.4 Results

We used ZoomShop to edit a variety of images with different composition goals. For each result, we show the original photo, the zoomed-and-cropped baseline, our modified image after applying a nonlinear $b(z)$, and inpainted final results. In cases where manually guided inpainting is necessary, we show both automatic and manual results side-by-side. We also show the view boundary used for each camera model (shown as top-down diagrams).

Our results are 2536 pixels wide and depth maps are 640 to 2048 pixels wide, with heights

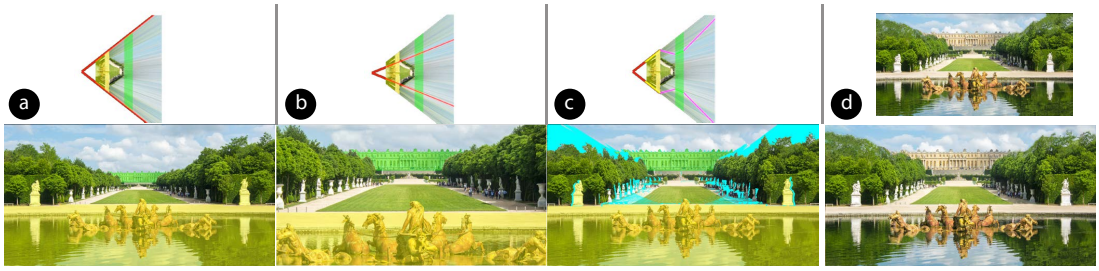


Figure 3.9: Versailles [96]. Goal: enlarge the Versailles palace while keeping the fountain fixed. a) Original photo. (b) Zoom in and crop. (c) Our result. (d) Our result with inpainting (top: automatic; bottom: with manual guidance). We use a linear interpolant and select depth ranges that border the start and end of the lawn. This hides bending artifacts where the perspective changes.

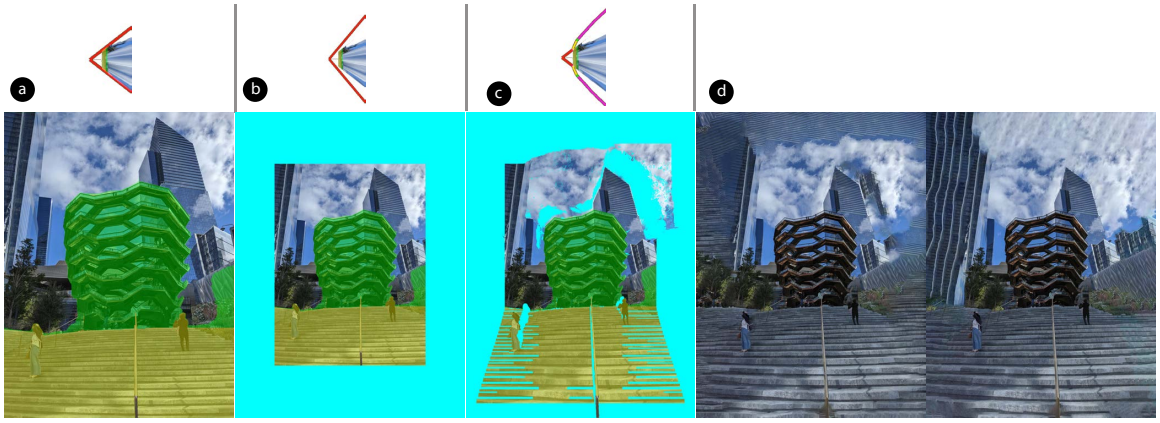


Figure 3.10: The Vessel. Goal: expand depth of the steps leading up to the Vessel. (a) Original photo. (b) Zooming out and cropping. (c) Our result. (d) Our result with inpainting (left: automatic; right: with manual guidance).

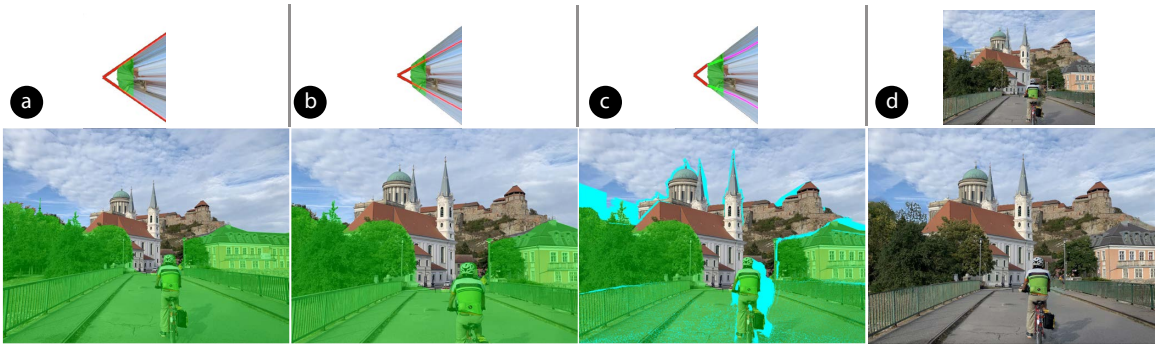


Figure 3.11: Biker Bridge. Goal: compress the depth of the bridge to better match our memory. (a) Original photo. (b) Zooming in and cropping. (c) Our result. (d) Our result with inpainting (top: automatic; bottom: with manual guidance).

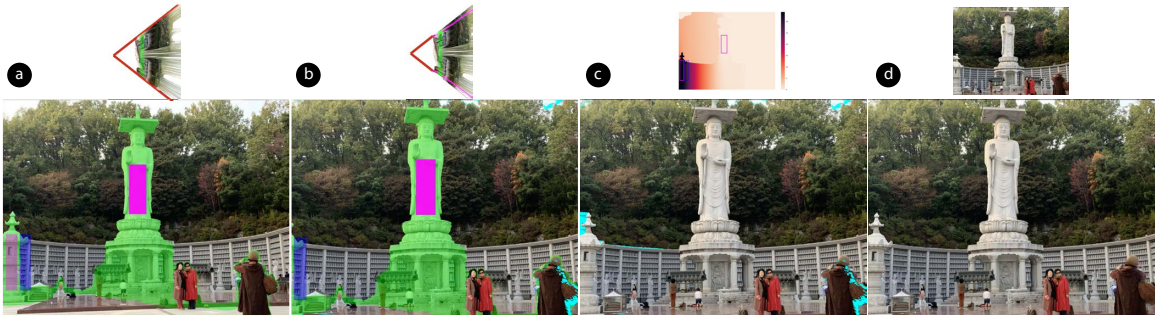


Figure 3.12: Buddha. Goal: enlarge the Buddha statue while keeping the left tower in view. Magenta and blue rectangles mark the input to our translation optimization. (a) Original photo. (b) Scaled up Buddha statue. (c) Our result (top: translation map). (d) Our result with inpainting (top: automatic; bottom: with manual guidance).

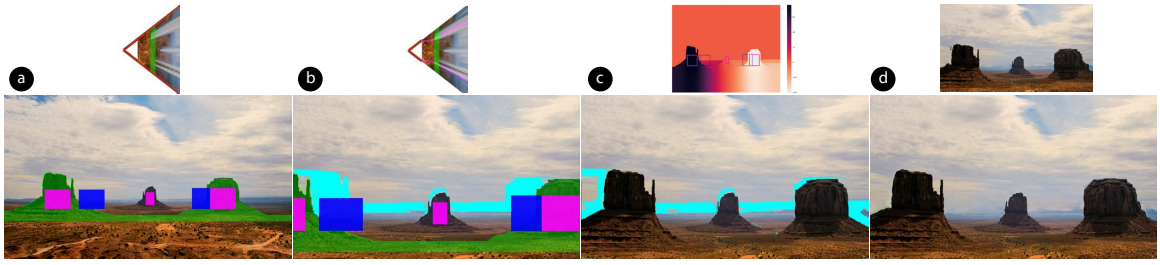


Figure 3.13: Monument Valley [136]. Goal: enlarge the hills, keeping all three fully visible. Magenta and blue rectangles mark the input to our translation optimization (each pair of rectangles has the same size; magenta overlays blue rectangles). (a) Original photo. (b) Scaled up hills. (c) Our result (top: translation map). (d) Our result with inpainting (top: automatic; bottom: with manual guidance).

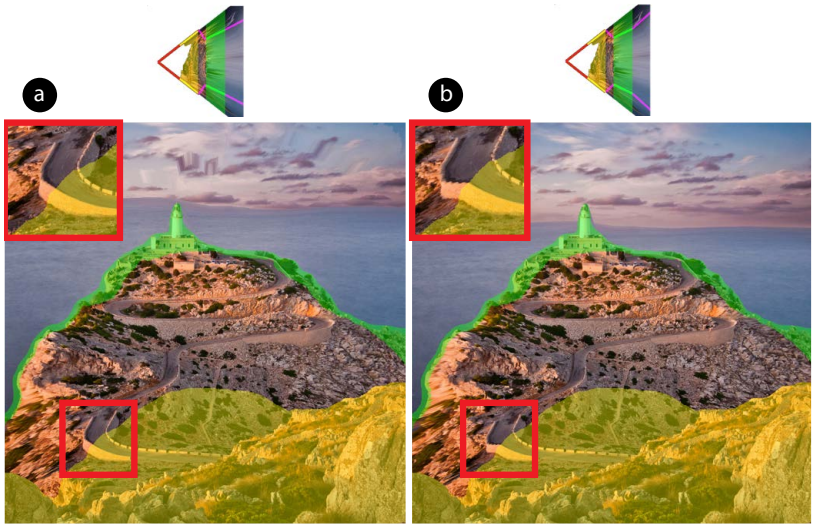


Figure 3.14: Lighthouse [97]. Two depth ranges are connected by (a) a curved (cubic Bézier) segment; (b) a linear segment. A linear segment yields an artifact at the far boundary of the yellow depth range, where the road bends suddenly (red boxes). A cubic segment smoothly varies the transition and avoids the artifact.

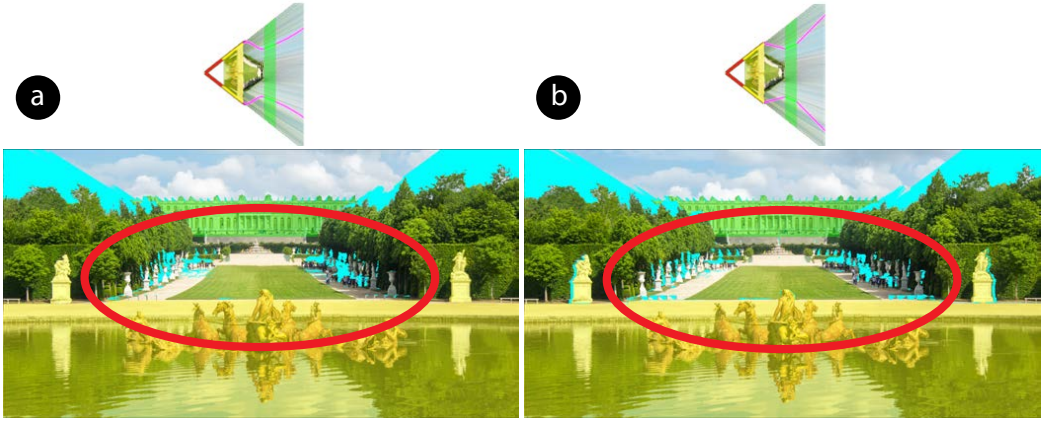


Figure 3.15: Versailles [96]. Two depth ranges are connected by (a) a curved (cubic Bézier) segment; (b) a linear segment. A curved segment does not preserve the straight lines of the lawn, creating a visible artifact (red circles). A linear segment preserves straight lines but can introduce a bend artifact. The artifact is hidden here because the segment boundaries are at the start and end of the lawn.

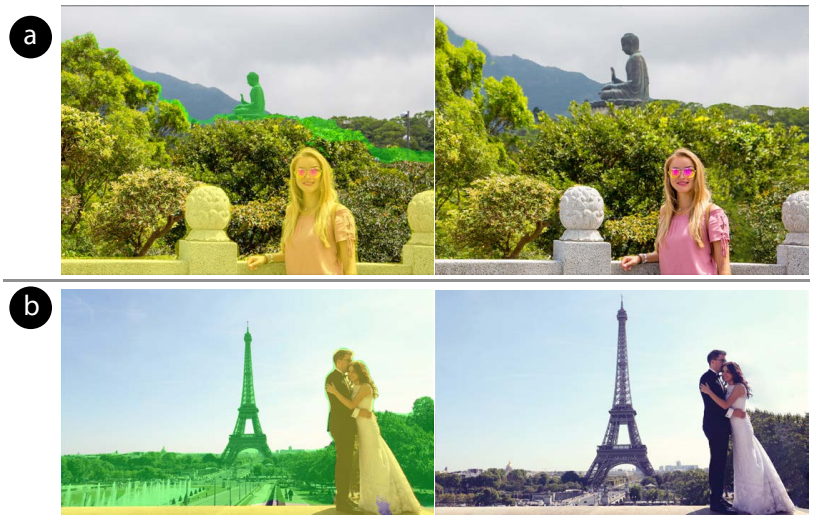


Figure 3.16: Tourism [67, 64]. (a) Goal: enlarge Big Buddha. (b) Goal: enlarge the Eiffel Tower. Left column: original photos. Right column: output images inpainted with manual guidance.

determined by aspect ratio. Manual clean-up of estimated disparity maps (by a novice user) takes 5 minutes to 3 hours. Scale and foreshortening edits are real-time with immediate feedback. The optimization for object translation takes 2 to 10 minutes. Inpainting with manual guidance takes 5 minutes to 1 hour. Although some manual effort is necessary, we focus on ZoomShop's core contribution of editing composition and rely on the rapidly advancing research for depth estimation and inpainting.

First, we focus on results where we adjust scaling. Figure 3.1 shows enlarging a boat while keeping foreground trees to maintain framing. Similarly, Figures 3.2 and 3.3 create balance by making important background objects (building, lighthouse) more prominent relative to less important foreground objects (boat, ridge). In Figures 3.7 and 3.8, the foreground objects establish the context of where each photo was taken from so these elements are preserved while making the primary subjects larger. In Figure 3.9 the background building is really much larger than the foreground fountain, so it is emphasized by enlarging it.

Next we adjust foreshortening. The structure in Figure 3.10 is diminished and the foreshortening of the stairs increased, expanding their depth, to create a more dramatic appearance. Conversely, in Figure 3.11, the foreshortening of the bridge is decreased, compressing its depth and making the scene feel closer.

Finally, we use translation to maintain visibility of important objects. In Figure 3.12 enlarging the statue pushes the tower out of the image, so it is smoothly translated with the connected wall back into view to restore framing. Multiple objects can also be translated, as in Figure 3.13, where two of the three hills get scaled out of the frame, and are both moved back towards the center with the ground in front smoothly varying.

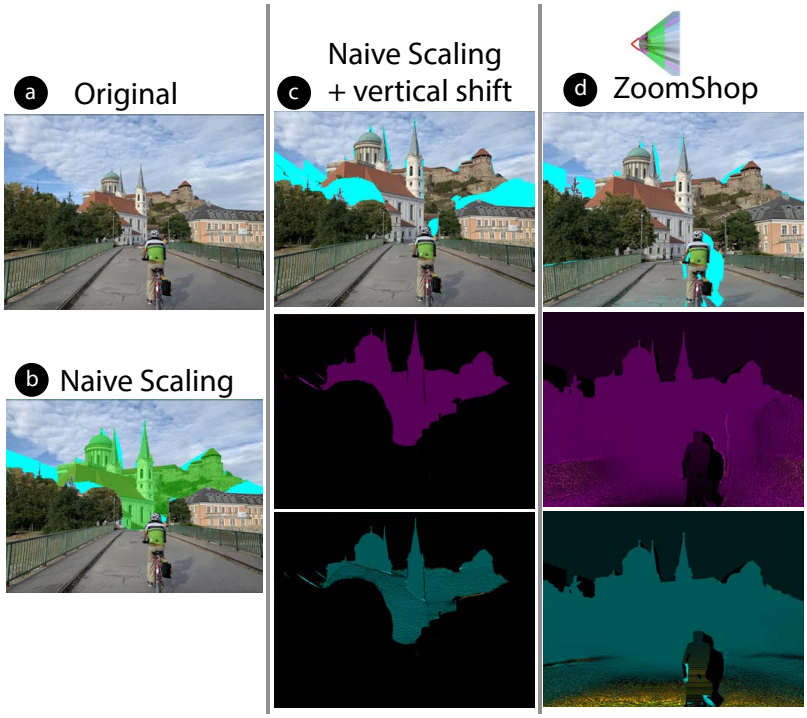


Figure 3.17: Biker bridge. Comparison with naively scaling depths independently. (a) Original photo. (b) Naively scaling the church breaks depth continuity and causes the church to become detached from the ground. (c) Users must take care to vertically shift the scaled region to reconnect to the ground. As shown in the horizontal (purple) and vertical (blue) scale maps, this naive approach only uniformly scales pixels of the church. (d) ZoomShop's camera model automatically accounts for depth continuity and prevents the church from detaching from the ground. As shown in the scale maps, ZoomShop scales the pixels of the ground in front of the church as well, to provide a smooth transition of scale across depth.

We use smooth interpolation for all results except for Figure 3.9, where smooth interpolation causes the lawn in front of the building to curve. In that case, we use linear interpolation and select depth ranges that border the start and end of the lawn to hide the perspective changes. Figures 3.14 and 3.15 give further examples of the impact of smooth vs. linear interpolation. Figure 3.16 shows other examples of tourism.

3.4.1 Comparisons to Alternatives

Scaling with ZoomShop's camera model is different from naively dividing the scene up into multiple depths and scaling each depth independently (Figure 3.17). Naively scaling scene elements can easily break depth continuity, whereas ZoomShop's camera model automatically accounts for it (unless the user intentionally chooses to break depth continuity by opting for a piecewise discontinuous model).

Niklaus et al. [103] create animations from an image by reconstructing the 3D scene and moving

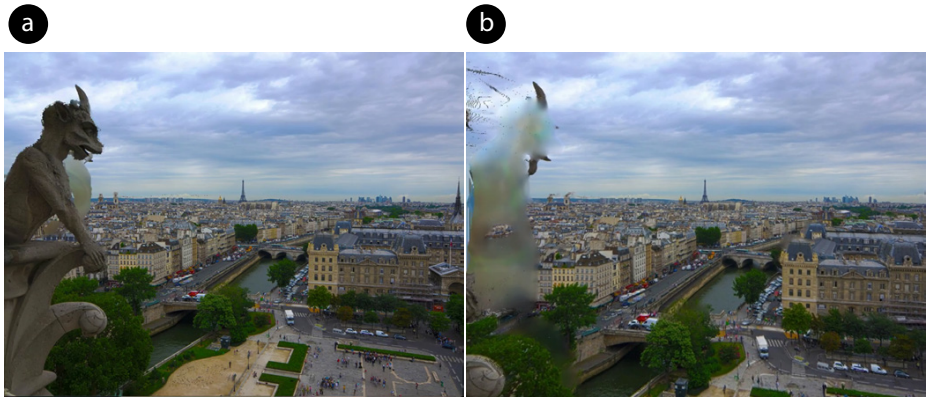


Figure 3.18: Eiffel Tower [92]. This result from Niklaus et al. [103] moves a virtual camera through the 3D scene creating a short animation. (a) The start of the animation. (b) The end of the animation. Because the camera uses linear perspective, it cannot enlarge the Eiffel Tower while keeping the gargoyle in view. Moving the camera forward makes the tower bigger but cuts out the gargoyle.

the camera through the scene. While their method also incorporates depth, their camera model uses linear perspective and thus cannot be used to scale foreground and background elements differently (Fig. 3.18). Since their goal is to add parallax in their animations, as opposed to photographic composition editing, they do not provide controls for adjusting object size, foreshortening, or position.

While our camera model is inspired by Computational Zoom [14], our model is more general and supports additional features that are significant to our results. First, ZoomShop supports non-monotonic boundary curves, i.e., segments of $b(z)$ with negative slope, which enables resizing background objects more dramatically (Fig. 3.19). Second, ZoomShop supports smooth interpolation of the boundary curve, which can be used to reduce visual artifacts in certain types of scenes (Fig. 3.14). Third, ZoomShop supports discontinuous boundary curves, which enables adjacent depths to be scaled very differently for greater artistic control (Fig. 3.7). Finally, ZoomShop can operate on a single image and depth map, whereas Computational Zoom requires acquiring multiple images to reconstruct the 3D scene, which would not be possible in may of our results due to physical constraints (Fig. 3.2) and scene motion (Fig. 3.11).

3.4.2 User Impressions

We asked two novice users to try out ZoomShop and give us feedback. Each user adjusted two images; one provided by us (Figure 3.3) and one they provided themselves. Before adjusting the images, they first stated their editing goals. Both users chose to enlarge some distant object while keeping some foreground elements fixed or reducing their size. In their feedback, both users claimed they achieved their photo editing goals and liked their results, except for artifacts due to depth

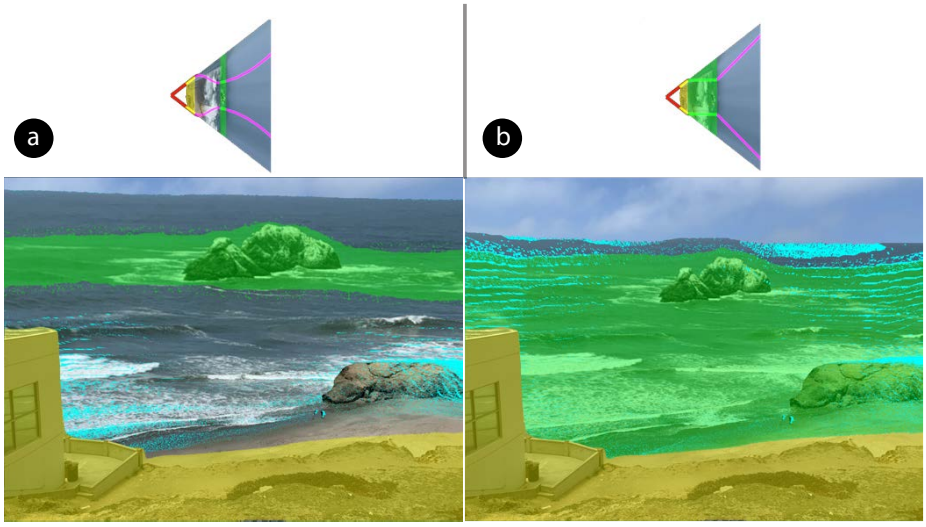


Figure 3.19: Seal Rocks. Comparison with Computational Zoom [14]. (a) ZoomShop's camera model supports non-monotonic boundary curves with negative slope segments, such as on the water between the yellow and green regions, which allows greater changes in scale. (b) Computational Zoom does not support negative slopes, limiting how large we can scale up Seal Rocks while keeping the Cliff House in place.

estimation issues, and assuming that the removed pixels (in cyan) will be inpainted well. One user commented that they often wished to achieve similar edits in their photographs and claimed that ZoomShop was "fun to play with," "really cool," "useful and easy to make a depth composite image," "much easier than Photoshop." Due to unfamiliarity with the controls, both users had minor hiccups when adjusting images but were able to achieve their goals after 1-3 attempts. One user gave some suggestions on using different keyboard mappings for the controls, but both claimed that the controls were "intuitive."

3.5 Discussion

For most of the results, it was necessary to manually clean up the disparity maps from MiDaS [113, 127] before using them as input to ZoomShop. Disparity values are inversely related to depth; for simplicity, we discuss our corrections in terms of depth instead of disparity. We include some examples before and after correction in Figure 3.20.

While metric-accurate depth is not necessary for ZoomShop to generate acceptable results, scene elements do need to have correct depth ordering. When two depths are scaled in different relative amounts under a new $b(z)$, any relative errors between the two depths become more pronounced after scaling and can lead to jarring inaccuracies in the scaled 3D scene. Example distortions include altered object borders (Figure 3.20a), stretched objects (Figure 3.20b), and wrong occlusions (Figure

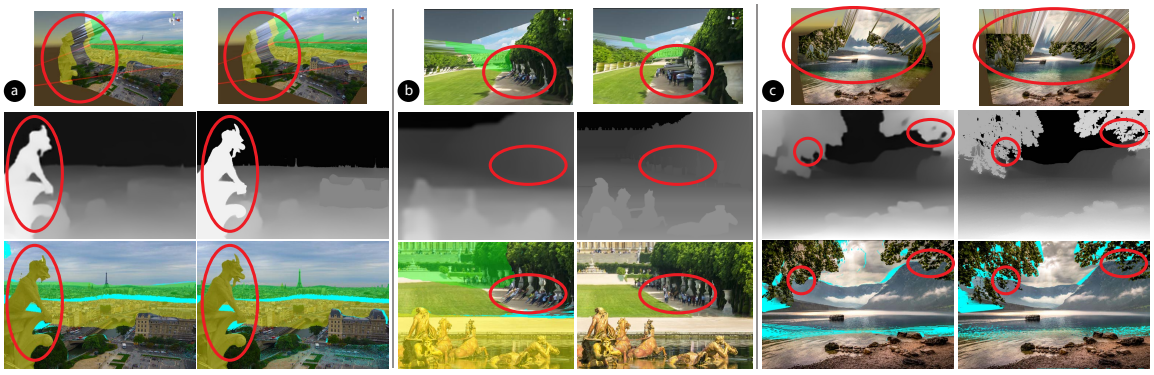


Figure 3.20: Correction of Depth Estimation Errors (left: before correction. right: after correction). (a) Blurry edges in Eiffel Tower [92]. Before correction, inaccurate depth estimation along the gargoyle's borders leads to depth bleeding in the 3D scene (top, left) and causes its borders to change after scaling (bottom, left). After correcting the depth inside the gargoyle, the gargoyle's outline is preserved (bottom, right). (b) Incorrect depth within scene elements in Versailles [96]. Before correction, the tourists received the same depth as the receding ground and became significantly distorted after scaling (bottom, left). After correction, the tourists remain vertical on the ground (bottom, right). (c) Missed fine features in Mountain Lake Trees [139]. The sky region between the leaves received the same depth as the leaves (top, left); after scaling, the sky incorrectly occludes the mountains (bottom, left). Fixing the depth of the sky between the leaves preserves depth order after scaling (bottom, right).

3.20c).

To fix these errors, we manually masked objects with incorrect depth ordering (e.g., gargoyle) using a combination of Photoshop's Lasso Tool and color range selection [2]. We then added a fill layer with the correct color for the masked region (e.g., fill entire gargoyle with the same color as the ledge it rests on). Because this process is time-consuming, the total time to make manual corrections depends on scene complexity (i.e., more regions to mask) and image resolution (i.e., mask boundaries need to be more accurate, and object outlines need to be more crisp).

For high-quality results, monocular depth estimation needs to produce correct depth ordering among scene elements, correct relative depth inside individual scene elements, and align depth boundaries with object edges. For photographs with humans, depth boundaries need to correctly align with the person's outline and may face more difficulty capturing fine hair. To avoid distortions of 3D face shapes, users can select the person inside a single depth range when making edits.

As mentioned in Section 3.3.5, Photoshop's CAF [2, 20] has limitations when automatically inpainting removed pixels. In general, CAF automatic inpainting works well for images whose missing pixels are only textured regions, but regions with more scene structure (e.g., the horizon, object boundaries) often require manually specifying regions for CAF to sample from (Figure 3.21) . For high-quality results, automatic inpainting should respect high-level scene structures in addition to extending textured regions.

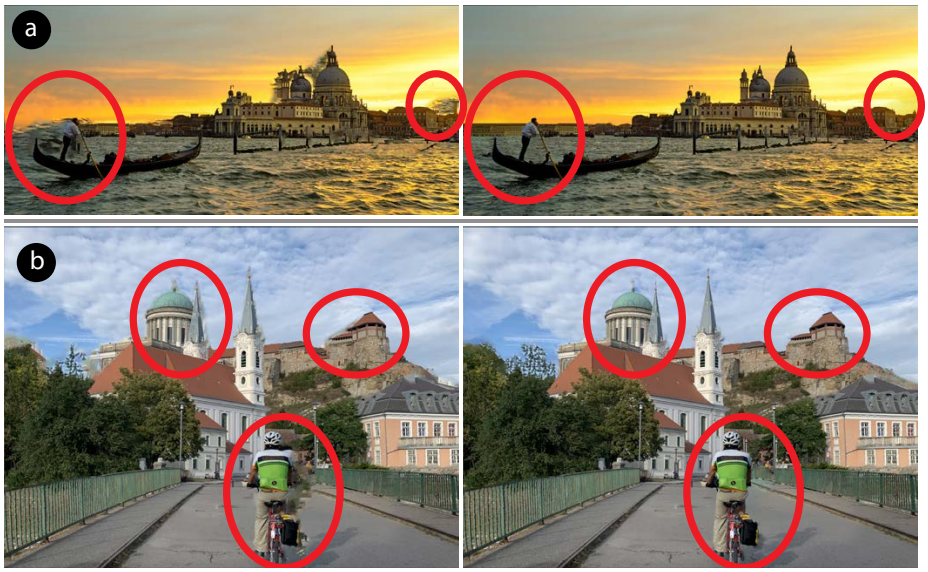


Figure 3.21: Inpainting: automatic (left) vs. manually guided (right). (a) Venice [137]. Automatic inpainting pasted some water texture where the sky should be. (b) Biker bridge. Automatic inpainting caused objects to "bleed" pixels beyond their boundaries. In both of these examples, we fixed these issues by manually specifying regions for CAF to sample from (e.g., sky).

Despite current limitations in depth estimation and inpainting, the extraordinarily rapid pace of recent progress [113, 127, 95, 146, 72, 79, 69, 135, 86, 160, 157, 91] suggest these areas will improve in the coming years. Our goal is to provide new, useful ways to edit photographs given these techniques.

3.6 Limitations and Future Work

Our non-linear camera model constrains all scene points at the same depth slice to scale by the same amount. While our optimization for object translation makes a step towards non-uniform scaling of scene points at the same depth, future work can investigate ways to make this feature more interactive and/or find other types of user control to non-uniformly scale depth. This could be useful if multiple objects are at the same depth, but the user would like to make some of them larger and more prominent than others.

ZoomShop does not automatically preserve straight lines in the image. Users can select depth ranges in regions where they want linear perspective (where straight lines are preserved) or use a linear interpolant. Future work could find ways to automatically preserve straight lines, regardless of the boundary curve.

ZoomShop lets users choose how to break up a scene by selecting depth ranges, how to piece the modified regions together using interpolants, and whether or not to break depth continuity in turn for greater changes in scale. While these features give users lots of flexibility to achieve a wide range

of compositions, future work can investigate ways to automatically determine salient depth ranges and recommend boundary curves to the user.

3.7 Chapter Summary

In this chapter, we focused on the task of manipulating relative object sizes, foreshortening, and positions in photographs and identified a set of task-dependent realism constraints for this task. We showed that the important properties to maintain (M) are depth structures, which include depth continuity and ordering. We also showed that we can relax (R) light ray properties to achieve a larger space of feasible solutions. Specifically, we demonstrated that even if light rays are not straight, we can still produce realistic-looking results for this task and achieve changes at larger scales than possible if we strictly adhered to those light ray properties. The input parameters (I) for this task are the selected pixels and amount of change (e.g., scale or offset).

We then presented ZoomShop, which applies the task-dependent realism constraints M and R as the user manipulates I and produces realistic-looking results. ZoomShop includes a non-linear camera model parameterized by the view boundary curve and a depth-aware image warp optimization to support object translation. These techniques enable a straightforward set of interactions for users to edit images by adjusting the appearance of objects without having to understand 3D scenes or camera models. We believe that ZoomShop increases the ways in which photographers can easily manipulate photographs to improve their expressiveness and better match their artistic intent.

Chapter 4

Gated Clips

In this chapter, we explore the task of adaptively extending the length (time) of 360° videos to satisfy gaze conditions. In 360° video, viewers have the freedom to look anywhere at any time; however, they may miss important events outside their field of view. To ensure that viewers see important events in 360° video, we introduce *gated clips*, which adaptively extend the length (time) of the video until a gaze condition is met, such as looking at a specific region of interest (ROI).

Our goal is to create gated clips with realistic extended visual content based on the viewer's direction. We identify a set of task-dependent realism constraints for gated clips and design new techniques to apply these constraints when generating gated clips. By applying task-dependent realism, we increase the space of realistic-looking solutions and automatically preserve important realism properties as the author specifies gates to achieve their storytelling goals.

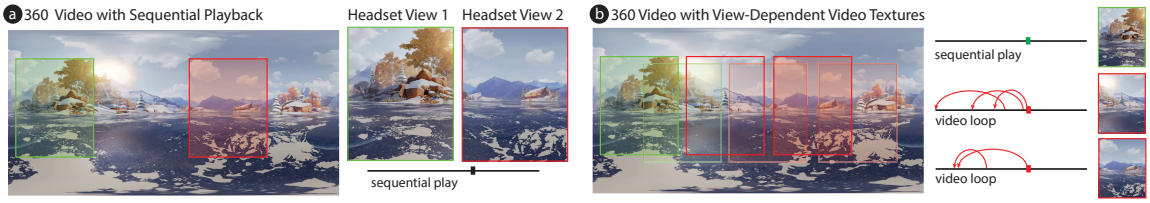


Figure 4.1: In 360° video, viewers can look anywhere at any time. In the opening scene of Invasion!, a rabbit emerges from a cave (a). In sequential playback, a viewer looking at the cave (green box) will see the rabbit emerge, whereas a viewer not looking at the cave (red box) will miss this event. We provide tools to guarantee that viewers see the region of interest (ROI) at the correct timecode to witness the event (b). We introduce the concept of gated clips, where playback only continues if the viewer satisfies a condition related to the ROI (green boxes). Otherwise, our player loops the video using view-dependent video textures (red boxes).

Figure 4.2: Our prototype desktop video editing interface with a *gated clip*. The upper left pane shows a preview of the headset FOV, which is output live to the Oculus Rift. The upper right pane shows the full equirectangular view and marks the current headset FOV (pink border). The video timeline on the bottom acts as a conventional video editor, with each dark blue rectangle representing a clip. The first clip is a gated clip (white border). Filmmakers specify a *gate timecode* (red vertical line on timeline), a *ROI* (green box) on the equirectangular frame, as well as other parameters shown in the settings box below the clip. View-dependent arcs are shown as backward arcs (red arrows) on top of the gated clip. In this example, the ROI is the aliens, and the *gate condition* is a *lookat gate*, i.e., playback may not advance past the gate timecode unless the viewer is looking at the ROI. To avoid static loops and reduce arcs with visual artifacts, the filmmaker can set thresholds on the length of arcs and on the perceptual difference of arc transitions. After setting all thresholds, the filmmaker can generate the view-dependent arcs by loading in (pre-processed) arc costs and applying the gate. To have viewers jump to the gate timecode as soon as they see the ROI, the filmmaker can also enable forward arcs that are under a perceptual threshold. Finally, the filmmaker can choose to cross-fade the audio during loop transitions or choose to mute the audio entirely.

4.1 Introduction

The medium of 360° video provides new artistic opportunities for filmmakers, allowing them to create videos with a greater sense of immersion and engagement than with conventional video. Yet, it also presents new challenges. In traditional cinematography, the director has full control over the camera orientation, field of view, zoom, and focus at all times. Traditional filmmakers use these controls to drive the narrative, ensuring that the viewer sees each important story element at the right time. With 360° videos, however, directors no longer have this control, and viewers can look in any direction at any time. As a result, viewers may miss important story content and become lost or confused as the story progresses.

For example, in the animated short "Invasion!" [19], the story begins with an establishing shot placing the viewer in the middle of an icy lake (Figure 4.1). Initially, the viewer is given time to look around and become familiar with their surroundings. A rabbit eventually emerges from a small cave. However, if the viewer is not looking at the cave entrance when the rabbit emerges, they will not see the rabbit, or what it does next. A second type of example is in the "Stranger Things: The VR Experience" [100] short film. As the tension rises, the viewer answers the phone, and is told to turn around. Then, a monster attacks from the direction opposite the phone. If the viewer does not turn around fast enough, they miss the monster attacking. A third type of example occurs in "Wild: The Experience" [44], where the viewer is placed between a hiker and an empty rock, on which a "ghost" appears only if the viewer is not looking at the rock. The hiker and the sound of her breathing is intended to get the viewer's attention away from the rock, so that the ghost can appear outside the viewer's field of view. However, if the viewer never looks away from the rock, the video player will reach the end of the video without the ghost ever appearing. Although the directors of these examples include passive gaze guidance techniques, such as audio cues, to encourage the viewer to look in a particular direction, none of the techniques are foolproof.

We propose a new filmmaking technique called *gated clips*, designed to ensure that a viewer sees key elements of the narrative in a 360° video. Using our technique, the filmmaker can author a *gate* which ensures that playback may only proceed past a *gate time* only if a filmmaker-defined viewer gaze condition is met, such as looking at a specific region of interest (ROI). For example, in the "Invasion!" short, we can place a gate just before the rabbit emerges from the cave and treat the gate ROI as the cave entrance. During playback, the gated clip would only proceed past the gate if the viewer is looking at the cave entrance. Otherwise, it would wait for the viewer by extending video content before the gate. Here, our goal is to generate gated clips with realistic extended content until the gate condition is met.

4.1.1 Task-Dependent Realism

In this task, the user-input parameters (I) are the gate time, a region of interest (ROI), and a viewing condition (e.g., look at the ROI). To preserve realism, the important properties to maintain (M) are within-view spatiotemporal coherency, i.e., the visual content within the viewer's field of view should look coherent in space and time.

In gated clips, we cannot play past the gate unless the viewing condition is met. Up until the gate time, the original clips have realistic visual content. However, if the gate condition is not met by the time video playback reaches the gate, we need to extend the clip length with realistic visual content until the condition is met. To achieve this, one of the properties which we can relax (R) is linear time – the timeline of the gated clip does not need to adhere to the originally captured linear timeline. Until the gate condition is met, we can allow the video to jump back to earlier frames and loop seamlessly while waiting for the viewer. Our approach is inspired by video textures [120], which play video frames out of order at places where it is unnoticeable for the viewer, thus extending a finite-duration video into an indefinitely long one.

Conventional video texture algorithms, however, are too constraining for 360° video, because they only allow transitions between frames when the change is imperceptible anywhere in the frame. In 360° video, since the entire view-sphere is encoded in a single equirectangular frame, it is difficult to find frames with imperceptible changes everywhere, due to the large amount of pixels. To combat this, our insight is that the change only needs to be imperceptible within the viewer's headset view. That is, the second property we can relax (R) is out-of-view spatiotemporal coherency; the visual content outside of the viewer's field of view does not need to be coherent. To this end, we introduce *view-dependent video textures* for 360° video, where the transitioned frames are selected based on the viewer's headset view at any given time. View-dependent video textures account for all possible head trajectories, so if the viewer moves their head (thus changing the content within their FOV), our technique ensures that there is at least one other frame to jump back in time to, such that spatiotemporal coherency is always maintained within the headset view.

By relaxing these constraints R, we consider a significantly larger amount of visual content and frame transitions, thereby increasing the space of realistic-looking gated clips. By applying task-dependent realism, we are able to formulate the generation of gated clips as an optimization problem and automatically maintain M as the user manipulates I.

4.1.2 Overview

We introduce new techniques to convert a standard 360° video clip into a gated clip with view-dependent video textures. Viewing such gated clips requires a new kind of video player that seamlessly loops the 360° video playback until the gate condition is met. To prototype these ideas, we present a user interface for editing 360° videos with gated clips (Figure 4.2). Our interface is built on a conventional timeline interface, but with special shot types for gated clips. We demonstrate

results on five different videos, four of them professionally produced and not intended for use with our technique. To understand viewer preferences, we ran a user study with standard and gated clips and found that 9 out of 11 users preferred videos with gating.

4.2 Types of Gates

As mentioned above, a gated clip is comprised of the following elements: The *gate timecode* is a specific frame in the clip. The video may only progress beyond the gate timecode when a *gate condition* is met. We use the term *timecode* to describe a frame index into the video timeline, and distinguish such timecodes from playback time, which may differ on each viewing due to looping.

We consider two types of gate conditions: (1) A *lookat gate* specifies that the viewer must see a specific region of interest (ROI) at the gate timecode for the video playback to proceed, i.e., that the ROI is within the viewer's field of view (Figures 4.1 and 4.2). (2) An *offscreen gate* is the inverse condition, i.e. it specifies that the viewer *must not* see the ROI at the gate timecode for video playback to proceed.

We give examples of three narrative use cases for these two types of gate conditions. A common trope in 360° filmmaking, such as in "Invasion!," (see Introduction section) is that the viewer is given a considerable amount of time to become familiar with a new environment before the first main action begins; this is akin to an establishing shot in conventional filmmaking. In such cases, authors could use a lookat gate when the viewer is initially placed in a new environment, with the gate timecode and ROI located at the time and spatial location where the main action begins. In the "Invasion!" example, the gate timecode would be placed at the time the rabbit first appears, and the ROI would be placed at the cave entrance. Another narrative use case is when the viewer is supposed to move their head in the middle of the story. For example, in "Stranger Things," the viewer is instructed to turn their head around in order to see the monster at the end of the hallway. In such cases, the author could again place a lookat gate right before the next action starts and set the ROI on the new target location. In this example, the ROI would be on the monster, and the gate timecode would be right before the monster attacks. Finally, for surprising entrances and disappearances, the author could use offscreen gates to ensure that the viewer does not see the actual appearance or disappearance. For example, in the "Wild" example, the author could place an offscreen gate right before the ghost appears and mark the rock as the ROI. That way, the viewer must look away for the ghost to appear offscreen, and then the viewer could turn to look at the rock and see the ghost.

Figure 4.3: Discretized view directions. This figure visualizes the FOVs of $|\mathbb{V}| = 6$ discretized view directions evenly spaced around the equator on an equirectangular frame from an example video. In our implementation, we used $|\mathbb{V}| = 40$. The horizontal FOV of each discretized view matches the horizontal FOV of the Oculus Rift and covers the full vertical range of the video.

4.3 View-Dependent Video Textures

A video texture is a video clip that can be played back endlessly by adding seamless transitions between non-sequential frames. Each transition occurs along an *arc* (t, t'), which transitions playback from frame t to frame t', over a user-specified cross-dissolve interval (our implementation uses a fixed 0.5 second cross-dissolve). By selecting arcs carefully, we can create seamless playback, in which ghosting is minimized. There are three types of arcs: *sequential arcs* $(t, t+1)$, used in normal playback; *backward arcs* $(t, t'), t' < t$, and *forward arcs* $(t, t'), t' > t+1$.

Constructing video textures normally involves finding *seamless arcs* (t, t') between non-sequential frames, such that cross-dissolving the video from frame t to frame t' is imperceptible to the viewer. The conventional approach is to measure some perceptual distance metric between the two frames. However, measuring the image distance for the entire 360° equirectangular image is too conservative, because viewers only see a small portion of the scene at any time; for example, typical VR headsets only have roughly an 80° horizontal field of view.

The core idea of view-dependent video textures is the use of *view-dependent arcs.* View-dependent arcs allow specific run-time transitions as long as the viewer is looking in a particular range of directions. These transitions are selected to minimize perceptual difference *within* the field of view. Specifically, we discretize the view-sphere with a fixed set of directions $v \in \mathbb{V}$. A view-dependent arc is then represented as a triplet (v, t, t') and is computed based on the pixels visible within v.

We chose a discretization of $|\mathbb{V}| = 40$ views. The views are equally-spaced around the equator (Figures 4.3 and 4.4). In our test videos, there is more motion around the equator and virtually

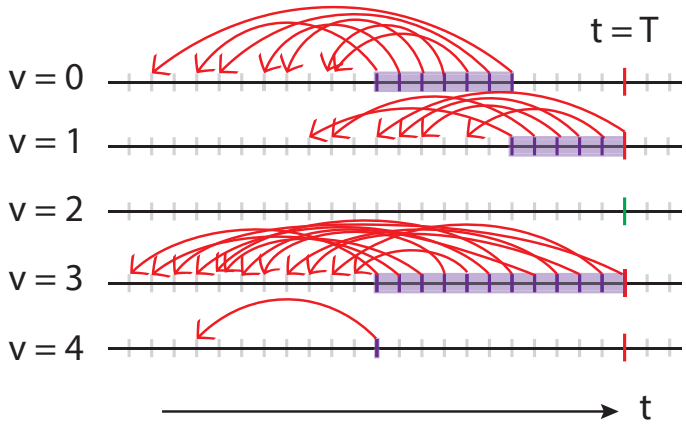


Figure 4.4: Visualization of view-dependent video textures for a gated clip. Each row corresponds to one of the discretized viewing directions (here we show the first five views only). The gate timecode is $t = T$, and $v = 2$ is the only view that satisfies the gate condition (green vertical line); other views do not (red vertical lines). Red arrows show the computed backward arcs. Tick marks correspond to frames, and purple ones are frames from which there are seamless backward arcs, i.e. arcs in which the transition frames have low perceptual difference.

no motion near the poles, so we chose the view FOV to have higher granularity of coverage in the horizontal direction than in the vertical direction. Thus, we set the horizontal FOV of each view to match the horizontal FOV of the Oculus Rift headset and the vertical FOV to cover the video's full vertical height ($w = 80.65°$, $h = 180°$). The FOVs of adjacent views overlap by 71.65°, with their centers 9° apart. Additional views could easily be added to the discretization, or coverage areas expanded as needed. See Discussion Section for more details on the trade-off between the number of discrete views and the FOV size of each view.

4.4 Generating View-Dependent Textures

Given a user-specified gate and an existing video clip, we wish to generate a view-dependent texture that satisfies the following properties: (1) The video proceeds past the gate timecode only if the viewer satisfies the gate condition. (2) The gate timecode is reachable if the viewer is looking in a direction that satisfies the gate condition (i.e., for lookat gates, in a direction where ROI is visible; for offscreen gates, in a direction where ROI is *not* visible). (3) The transitions taken along arcs minimize or eliminate ghosting. (4) All arcs satisfy user-set thresholds on the length of the arcs and on the perceptual difference of arc transitions. In order to discourage repetitious or static loops, the arc length threshold requires all backward arcs to be longer than a minimum threshold duration. To minimize ghosting and visual artifacts, our tool allows authors to specify a perceptual threshold, which is the highest perceptual difference of frames that arcs can have in order to be considered

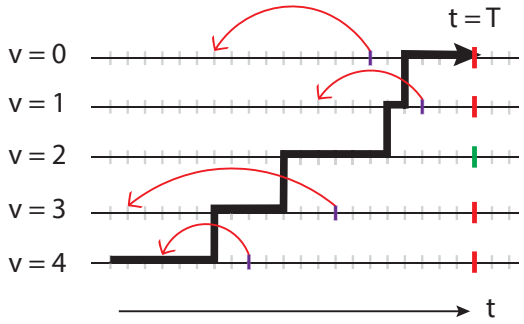


Figure 4.5: One naive solution is to find one loop (i.e., one backward transition arc) for each view that does not satisfy the gate condition. In this example, the gate timecode is $t = T$, and the view $v = 2$ satisfies the gate condition. All other views include one backward arc forming a loop. The thick black line shows an example viewer's head trajectory through the views over time (slowly turning their head from $v = 4$ to $v = 0$). Unfortunately the viewer can still get past the gate time through another view $v \neq 2$ via certain head motions, so the naive solution does not provide a guarantee that the gate condition is met before playback progresses.

seamless arcs.

Given these constraints, our goal is to specify the playback behavior for each combination of frame t and discretized view direction v. There are only two possibilities from each (v, t): either play forward to $t + 1$, or transition backward to some previous timecode $t' < t$. Our algorithm outputs the action (play forward to next frame or transition backward to an earlier frame) to take from each (v, t). Then, during playback, our player looks up the run-time frame t and the nearest discretized v, and follows the selected behavior at (v, t).

A naive solution is to create a separate video loop (i.e., one backward transition arc) for each view direction, except for those that satisfy the gate condition. That is, for each view direction $v \in \mathbb{V}$, we could independently search for a backward arc (v, t, t') that minimizes ghosting. Unfortunately, this approach does not guarantee that the viewer satisfies the gate condition. As illustrated in Figure 4.5, it is possible for viewers to move their heads in a way that allows them to pass the gate timecode through a view that does not satisfy the gate condition. We also considered a version of this approach which finds a single timecode t for all backward arcs (with independent destination timecodes t'), but this approach similarly does not guarantee that viewers satisfy the gate condition. If t is earlier than the gate timecode, viewers can still get past the gate timecode T through a view that does not satisfy the gate condition (Figure 4.6). Requiring t to occur at the gate timecode is too restrictive to work for general videos. For example, if a view (not satisfying the gate condition) is static for all frames $t < T$, but an object in the view moved at $t = T$, then the view would not have any seamless backward arcs originating at $t = T$, even though there are many pairs of frames before T that can form seamless arcs.

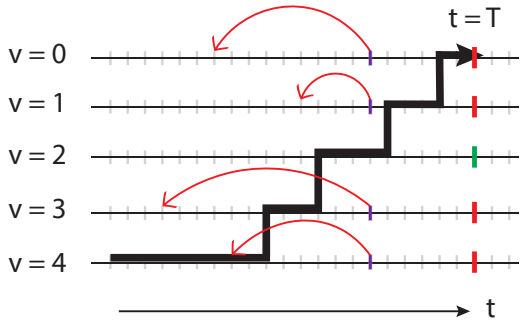


Figure 4.6: Another naive solution is to find one loop (i.e. one backward transition arc) for each view that does not satisfy the gate condition, such that all the arcs originate at the same timecode. In this example, the gate timecode is $t = T$, and the view $v = 2$ satisfies the gate condition. Unfortunately, if the originating timecode for all arcs does not equal the gate timecode T, then the viewer can still get past the gate time through a view $v \neq 2$, as shown by the thick black line, which gives an example of viewer's head trajectory (slowly turning their head from $v = 4$ to $v = 0$) that gets past the gate without satisfying the gate condition.

4.4.1 Graph Cut Formulation

We formulate the problem in terms of graph theory, specifically an s-t graph cut [47, 121, 23]. The graph construction represents playback as a state machine, but with some modifications, so that a minimal graph cut produces a solution to our problem of generating view-dependent video textures for gated clips.

The graph includes one node (v, t) for each pair of view direction and frame in the clip, from the start frame 0 to frame $T + 1$, which is the frame immediately after the gate timecode. Let $v \in \mathbb{H}$ be the set of views that satisfy the gate condition. For lookat gates, $\mathbb{H}$ is the set of views in which the ROI is visible; for offscreen gates, $\mathbb{H}$ is the set of views in which the ROI is *not* visible. We call (v, T), where $v \in \mathbb{H}$, *gate nodes*.

Graph Partition. Our goal is to partition the graph into two parts, a "safe zone," which the viewer must stay within before they satisfy the gate condition, and an "unsafe zone" that the viewer must *not* visit before the gate condition is met. The safe zone must include all nodes at the starting frame $(v, 0)$ as well as the gate nodes, because the viewer can start in any view direction, and the viewer must be able to visit the gate nodes in order to satisfy the gate condition. The unsafe zone must include nodes $(v, T+1)$ for all v, because viewers should not visit the frame immediately after the gate timecode if the gate condition has not been satisfied. With this construction, the viewer can exit the safe zone of the graph only by passing through a gate node. Otherwise, for boundary nodes in the safe zone that border the unsafe zone, our graph cut algorithm finds seamless backward arcs from which to transition back in time, so that the viewer does not enter the unsafe zone. If the viewer is at a node in the unsafe zone before the gate condition is met, they might see ghosting

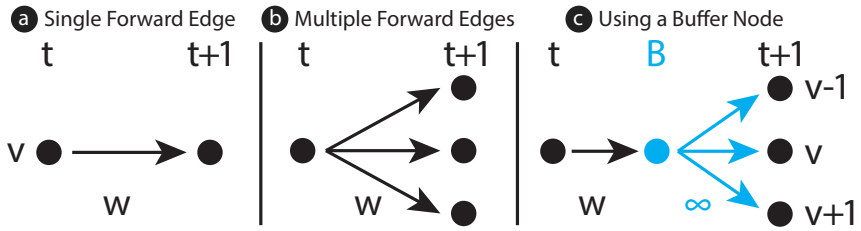


Figure 4.7: (a) A single forward edge between consecutive nodes $(v,t) \rightarrow (v,t+1)$ represents normal video playback of sequential frames but does not model head rotation. (b) Adding forward edges to adjacent viewing directions models head rotation below some velocity—i.e. $(v,t) \rightarrow (v',t+1)$, for all $v' \in \mathbb{N}(v)$. But cutting these edges corresponds to disallowing some head motions, which we cannot control. (c) A buffer node $(v,t)_B$ (cyan) creates one edge (black) that can be cut (to remove the video frame advance for this view), while infinite-weight edges (cyan) cannot be cut (to properly model free user head rotation). The edge weight w is designed to strongly prefer arcs with perceptual error below the user-set threshold. More details of how w is determined is in the Appendix.

and/or pass through the gate timecode in a view $v \notin \mathbb{H}$.

Accounting for Head Motion. At any instant, the viewer may rotate their head. Hence, from any node (v,t), the view direction at the next time instant may be from a neighbor set $\mathbb{N}(v)$, determined as a function of the field of view, the view discretization, and how fast viewers typically rotate their heads. Based on the work of Bussone [28], we assume a typical maximum head velocity movement of 9.03 rad/s. For $|\mathbb{V}| = 40$ and 30fps video, this means that viewers can move across $n = 2$ adjacent views in either direction over the course of one frame interval, and therefore the neighbor set $\mathbb{N}(v)$, of view v contains 5 views including v.

One might imagine representing sequential playback with allowance for head motion by including an edge from each node $(v,t) \rightarrow (v',t+1)$, for all $v' \in \mathbb{N}(v)$. However, this approach would allow the graph cut algorithm to cut some of these edges and not others, which would correspond to allowing some head movements and not others. Since we cannot control the viewer's head movements, we cannot use such a representation. Hence, we introduce *buffer nodes* $(v,t)_B$ between consecutive nodes (Figure 4.7). From each node (v,t), we insert an edge to its buffer node $(v,t)_B$, and, from the buffer node, we add edges to the subsequent nodes $(v',t+1)$ for $v' \in \mathbb{N}(v)$. The edges $(v,t) \rightarrow (v,t)_B$ are called *buffer edges*.

Cutting a buffer edge indicates that, in the output video texture, the corresponding sequential arc $(v,t) \rightarrow (v,t+1)$ is omitted from the video texture, and that a backward arc must be taken whenever (v,t) is reached.

Edge Weights. The weight of a buffer edge $(v,t) \rightarrow (v,t)_B$ depends on the best backward arc available from the node (v,t), since some backward arcs may introduce more ghosting than others. The buffer edge weight is designed to strongly prefer arcs with perceptual error below the user-set threshold. Details of how we determine buffer edge weights are given in the Appendix. Edges

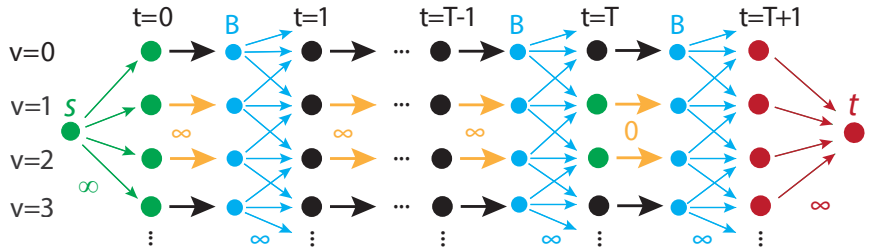


Figure 4.8: Our graph representation of a gated clip. The gate frame is $t = T$ and the views that satisfy the gate condition are $\mathbb{H} = \{1, 2\}$. We add buffer nodes (cyan) between consecutive frame nodes. There are infinite-weighted edges from each buffer node to possible views that viewers might see at the next frame. In this figure, $n = 1$ for the number of adjacent views that viewers can visit in one frame, but in our result videos we used $n = 2$. Buffer edges connect each frame node to its buffer node. Buffer edges of $\{(v, t)|v \in \mathbb{H}, t < T\}$ have infinite weight, while buffer edges of gate notes $\{(v, T)|v \in \mathbb{H}\}$ have a weight of 0. Source node $\boldsymbol{s}$ is connected to the starting frame nodes $t = 0$ in each view, while terminal node $\boldsymbol{t}$ is connected to all nodes at $t = T + 1$. After performing graph cut, the initial nodes and gate nodes (green) are partitioned from end nodes (red).

connecting buffer nodes to subsequent nodes $(v, t)_B \rightarrow (v', t + 1)$ for $v' \in \mathbb{N}(v)$ represent the set of views a viewer might transition into due to head movement. Since we cannot control viewer's head motion, these edges should not be cut, so we assign them a weight of infinity.

If the viewer is looking in a direction $v \in \mathbb{H}$ that satisfies the gate condition, they should be able to reach the gate node (v, T). In other words, all nodes (v, t), where $v \in \mathbb{H}$, $t < T$, should play forward (i.e., not traverse backward arcs). Thus, the buffer edges of these nodes should not be cut, so we set their weights to infinity.

The *s-t* graph cut formulation involves a source node **s**, which we connect with infinite-weight edges to the nodes that must be in the safe zone, i.e., initial nodes $(v, 0)$ for all view directions $v \in \mathbb{V}$. We do not explicitly connect **s** to gate nodes, because they are guaranteed to be partitioned into the safe zone due to the infinite-weight buffer edges $(v, t) \rightarrow (v, t)_B, v \in \mathbb{H}, t < T$. The sink node **t** is connected with infinite-weight edges to all nodes $(v, T + 1)$, which must be in the unsafe zone. The complete gated clip graph is shown in Figure 4.8.

In order to perform the s-t graph cut, we need at least one buffer edge in each $v \in \mathbb{H}$ to have non-infinite weight, so we set buffer edge weights of gate nodes $(v, T) \rightarrow (v, T)_B$ to 0. Consequently, edges $(v, T) \rightarrow (v, T)_B$ are always cut for $v \in \mathbb{H}$. Normally, cutting a buffer edge $(v, t) \rightarrow (v, t)_B$ indicates that a backward arc must be taken whenever (v, t) is reached. However, when the viewer reaches one of the gate nodes (v, T), where $(v, T) \rightarrow (v, T)_B$ is cut, we simply keep playing forward to pass the gate, instead of taking a backward arc.

Properties of Cut. The graph-cut algorithm solves for the set of buffer edges to remove with minimum total cost, such that the sink node **t** is not reachable from the source node **s**. This partition corresponds to segmenting the graph into a safe zone, including the start nodes and the gate nodes, and an unsafe zone, which include paths that violate the gate condition. Our implementation uses

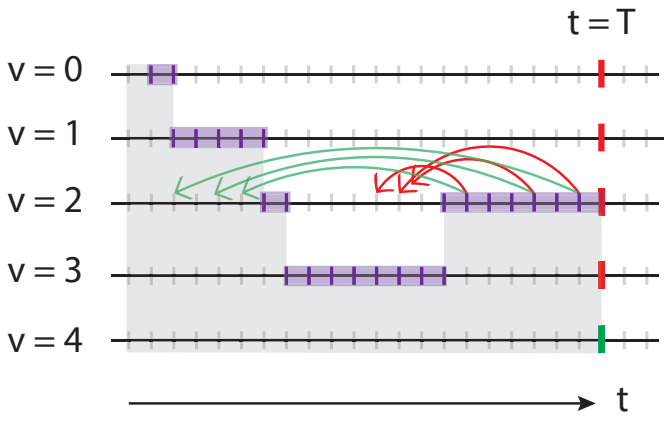


Figure 4.9: The graph cut algorithm may inadvertently create backward arcs (red arrows) into the "unsafe zone" when there are discontiguous cuts along a viewing direction. Our post-processing step replaces those backward arcs with new backward arcs (green arrows) that terminate in the "safe zone." The safe zone includes the both gray and purple shaded regions. However, since the purple frames border the unsafe zone, it is possible for viewers to turn their head into the unsafe zone as the player transitions into a purple frame, so our heuristic looks for replacement arcs that end in the gray shaded area.

the min-cut solver of Boykov and Kolmogorov [23] and takes an average of 0.14 seconds to compute the cut for the clips we have tested (Table 4.1).

4.4.2 Postprocessing

Once we run graph-cut on the graph, our video player extracts from the resulting partition a binary decision for each node: whether to (1) play forward sequentially from that frame or to (2) take a backward arc from that frame. If a buffer edge $(v,t) \rightarrow (v,t)_B$ is *not* cut, the video player plays forward sequentially from (v,t). As discussed in the Edge Weights section, the weight of a buffer edge $(v,t) \rightarrow (v,t)_B$ depends on the best backward arc available from the node (v,t). So if a buffer edge $(v,t) \rightarrow (v,t)_B$ is cut, then the video player traverses the best backward arc from (v,t).

It is possible for our graph cut algorithm to cut edges and produce backward arcs that end in the unsafe zone, i.e., a node that the viewer is not meant to reach before satisfying the gate condition. As shown in Figure 4.9, when there are discontiguous cuts along a viewing direction, it is possible for some backward arcs in that view to end on a node which is partitioned into the unsafe zone. We use the following heuristic post-process to correct these cases. We first identify the origin timecode t_C of the earliest backward arc along view direction v ($t_C = \min_{\{(v,t,t')\}} t$). We are guaranteed that all nodes before this time are in the "safe zone." Thus, we replace each backward arc (v,t,t') that ends after t_C ($t' > t_C$), with the best backward arc from (v,t) that ends before t_C. This approach

can produce video textures with some poor backward arcs; we highlight such poor backward arcs in the user interface, and the user may perform further adjustments to generate better results, i.e., re-run the algorithm with different parameters (e.g., gate timecode, ROI, arc length) .

4.4.3 Forward Arcs to the Gate

Our graph-cut algorithm creates backward arcs in views that do not satisfy the gate condition, i.e., $v \notin \mathbb{H}$. If the viewer looks at a view $v \in \mathbb{H}$, by default our player just plays normally until the viewer reaches the gate timecode. However, this may take some time, and the filmmaker may want the viewer to get to the gate timecode as soon as they look in the right direction. Thus, we provide filmmakers with the option to add forward arcs to views v that satisfy the gate condition. The forward arcs allow playback to jump directly to the gate when the viewer is in a view $v \in \mathbb{H}$ without forcing them to wait for the remaining duration of the gated clip. If the author selects this option, our system automatically adds forward arcs to all nodes $(v,t), v \in \mathbb{H}, t < T - 0.5$ sec for which the transition cost from (v,t) to $(v, T-0.5 \text{ sec})$ is below a user-set perceptual threshold. We set forward arcs to jump to 0.5 seconds before the gate timecode to allow the transition cross-dissolve to finish by the time the viewer gets to the gate.

4.4.4 Alternative Q-Learning Approach

To avoid producing backward arcs that end in the unsafe zone (e.g., Figure 4.9), we present an alternative approach to graph cut. In this case, the graph construction represents playback as a state machine, and we use Q-learning [147] to find the best action to take from each state during playback. While this Q-learning approach will find a set of backward arcs that end in the safe zone (if such a set exists), it requires iterating until convergence and thus takes longer than the graph-cut solution.

The graph includes one node (v,t) for each pair of view direction and timecode in the clip, from the start timecode 1 to $T+1$, one frame after the gate timecode. Each node represents a state that the viewer can be in during playback. Let $v \in H$ be the set of views that satisfy the gate condition, i.e., the ROI is visible or not visible, for lookat and offscreen gates, respectively.

At each state (node), our tool selects an action $a \in A$, where A is a set of all possible actions, and transitions to a subsequent state (node). The set of possible actions includes jumping to a previous frame or continuing to play forward (Figure 4.10). We denote the destination frame of the action as t_a, so if the current state (node) is (v,t), then $t_a \in [1...t-1, t+1]$. Our goal is to pass the gate through one of the *gate nodes* (v,T), where $v \in H$, and avoid getting pass through one of the *non-gate nodes* (v,T), where $v \notin H$. Once we pass the gate and arrive at $(v,T+1)$, we no longer want to jump back in time. Since we only build our nodes up to column $f = T+1$, the only action our tool takes from state $(v,T+1)$ is back to itself, i.e. $t_a = T+1$.

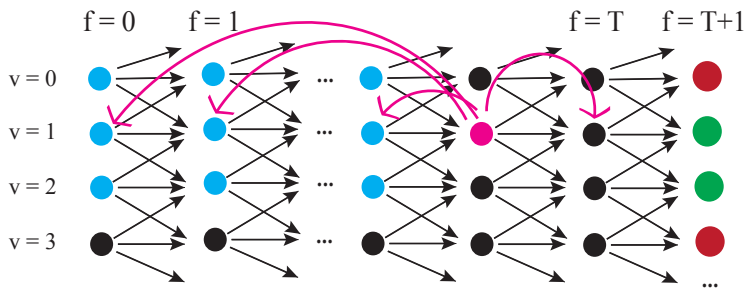


Figure 4.10: Our graph representation of a gated clip. The gate frame is $t = T$ and the views that satisfy the gate condition are $H = \{1, 2\}$. In this figure, $n = 1$ for the number of adjacent views that viewers can visit in one frame, but in our result videos we used $n = 2$. At any given node, our tool takes an action to either go to a previous frame or to continue playing forward. In this figure, the pink node is the current node, and each pink arc represents an action our tool can take from the pink node. Due to head motion, the viewer might actually end up $\pm n$ views within the destination view (highlighted in cyan).

At any instant, the viewer may rotate their head. Hence, we use a transition function to model head motion. Given a state-action pair $((v, t), t_a)$, where (v, t) is the current node and t_a is destination frame, $T(v, t, v', t_a)$ is the probability that the viewer reaches node (v', t_a) as the next state. The transition from t to t_a is deterministic (we guarantee a jump to frame t_a). The resulting v', however, is *not* deterministic due to head motion. We assume a typical maximum head velocity movement of 9.03 rad/s. For $|V| = 40$ and 30fps video, this means that viewers can move across $n = 2$ adjacent views in either direction over the course of one frame, and there are 5 views in the neighbor set $N(v)$, including v. Thus, $v' \in N(v)$.

We assume that the viewer stays in the current view 80% of the time, and turns to another view $v' \in N(v), v' \neq v$, 20% of the time.

$$T(v, t, v', t_a) = \begin{cases} 0.8, & v' = v \\ \dfrac{0.2}{2n}, & v' \neq v \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

For each transition, we assign a reward value to encourage reaching gate nodes and discourage reaching non-gate nodes. When the viewer is looking at a view that satisfies the gate condition, i.e. nodes $v \in H$, we only want to play forward, so we assign $-\infty$ to backward transitions from those views. For backward transitions from other views, we assign a reward to prefer transitions with perceptual error below a user set threshold.

$$R(v,t,v',t_a) = \begin{cases} 0, & \text{if } t_a = t+1 \text{ and } t_a \neq T+1 \\ 0, & t = t_a = T+1 \\ 100, & v' \in H, t = T, t_a = T+1 \\ -\infty, & v' \notin H, t = T, t_a = T+1 \\ -\infty, & v \in H, t_a < t \\ 0, & \text{if } t_a \leq t - M \text{ and } C(t,t_a,v') < \gamma \\ -C(t,t_a,v'), & \text{otherwise} \end{cases} \tag{4.2}$$

where M is the minimum loop length, and γ is the cost threshold, both defined by the user.

We then perform sample-based Q-learning on this model. Specifically, for each possible state-action pair, we want to find the optimal $Q^*((v,t),t_a)$ value, which is the expected sum of rewards for jumping to t_a at node (v,t) and thereafter continuing to select jumps t_a^* that maximize expected rewards.

$$Q^*((v_0,t_0),t_{a_0}) = E[\sum_{i=0}^{\infty} \beta^i R(v_i,t_i,v_{i+1},t_{a_i}^*)|_{(v_0,t_0),t_{a_0}\ldots t_{a_{i-1}}^*}] \tag{4.3}$$

where $t_i = t_{a_{i-1}}$ for $i > 0$ and $\beta \in [0,1]$ is the discount factor for future rewards. Recursively, Q^* is defined as:

$$Q^*((v,t),t_a) = \sum_{v'} T(v,t,v',t_a)[R(v,t,v',t_a) + \beta \max_{t_a'} Q^*((v',t_a),t_a')] \tag{4.4}$$

We approximate $Q^*((v,t),t_a)$ through a series of iterative updates, which is guaranteed to converge. At each iteration i, we compute Q_i for all state-action pairs $((v,t),t_a)$, which represents the Q-value at depth i, i.e. the Q-value of $((v,t),t_a)$ if we only take a total of i steps. Thus, we iteratively update the Q-values of all state-action pairs with:

$$Q_i((v,t),t_a) = (1-\alpha)Q_{i-1}((v,t),t_a) + \alpha[R(v,t,v',t_a) + \beta \max_{t_a'} Q_{i-1}((v',t_a),t_a')] \tag{4.5}$$

where α is the learning rate. We use $\alpha = 0.9$ and $\beta = 0.9$. The subsequent state (v',t_a) from each state-action pair is obtained by sampling $T(v,t,v',t_a)$.

We keep updating the Q values until they converge, i.e. when $|Q_{i+1} - Q_i| < \epsilon$ for some small ϵ. Then, we extract the optimal policy π of actions to take at each node:

$$\pi(v,t) = \arg\max_{t_a} Q((v,t),t_a) \tag{4.6}$$

Figure 4.11: Stills from our example videos. The first four were professionally-created videos, not intended for use with gated clips. The fifth we shot for this project.

4.5 Example Videos

We used our editing tool to add gated clips to five 360° videos (Figure 4.11). We selected a range of video genres and scenarios and varied our gate types to demonstrate a range of narrative use cases, detailed in Table 4.1. We created four videos (Invasion!, Stranger Things, Wild, and Lions) based on existing, professional videos, and shot the fifth video (Murder Mystery) ourselves. Note that the professional videos were not shot with gating in mind; we added gating in order to demonstrate our method. We cut each video down to one or two minutes in length. For each video we created three gated clips and made sure to place the gate at important story events.

We authored audio manually for the gated clips. By default, we cross-dissolved the audio during transitions, just as we cross-dissolved the video; we used this approach for "Stranger Things," "Wild," and "Murder Mystery." For "Lions," the narrator sometimes speaks during a gated clip. To avoid looping the narration, we played the audio normally (without transitions), separate from the visual content which may be looping. If the gated clip audio ended before the viewer passed the gate, we paused the audio until the viewer did, after which we resumed audio with the next clip. For "Invasion!", the original soundtrack includes music; we found that audio dissolves were jarring, so we muted the audio entirely.

Invasion! [19]**.** We added lookat gates to focus the user's attention at three key moments: the rabbit's entrance in the opening scene, the aliens' comedic entrance from the spaceship, and the aliens' attempt to attack the rabbit. The lookat gates help pace the story as the viewer looks back and forth between the rabbit and the aliens.

Video	Genre	Length (sec), Type					
		1^{st}		2^{nd}		3^{rd}	
Invasion!	Comedy	7	L	7	L	4	L
Lions	Docu.	7	L	4	L	5	L
Stranger Things	Horror	4	L	5	L	4	L
Wild	Drama	10	L	3	L	7	O*
Murder Mystery	Mystery	6	L*	4	O*	7	L*

Table 4.1: Summary of example videos. For each video, we added three gated clips. We show the length (in seconds) and type of each gated clip. L: lookat gate, O: offscreen gate, *: enabled forward arcs.

Stranger Things [100]**.** In this video, the viewer starts out in the living room. The camera then automatically moves first towards the dining room, and then in an opposite direction down a hallway. We used a lookat gate to ensure that viewers look at the dining room and down the hallway before the camera starts moving, so that they are looking in the direction they move towards. Otherwise, the unanticipated camera motion could be confusing and disorienting. We used a lookat gate to ensure that the viewer turns around before the monster attacks the viewer.

Wild [44]**.** In this video, a hiker rests on a rock and sees the "ghost" of her mother, who appears and disappears opposite the hiker. The viewer must look back and forth to see one and then the other. We added two lookat gates; one for the hiker when the ghost appears, so that viewers see the main character and do not witness the ghost's appearance, and subsequently one for the ghost. Finally, we added an offscreen gate with forward arcs, so the ghost immediately disappears when the viewer looks away.

National Geographic Lions [99]**.** In this documentary, the narrator occasionally refers to specific lions within a group, who each briefly become the main character. We added lookat gates to wait for the viewer to look at the correct lion before allowing the narration for that lion to begin. In addition, we added a lookat gate right before a lion attacked another lion, to ensure viewers see this important action.

Murder Mystery. This video is similar to Wild, in that a ghost appears opposite from the main character (with the viewer in-between the characters) and disappears when the character looks away; however, there is more background motion which makes looping more difficult. We add a lookat and an offscreen gate for the ghost's appearance and disappearance, as well as another lookat gate for the position where the ghost was standing, so the viewer sees that the ghost has vanished.

Our source code and the gated clip metadata used to produce these examples are available at the project website: https://lseancs.github.io/viewdepvrtextures/

4.6 User Study

In order to understand the effects of gated clips, we asked viewers to watch the five videos described in the previous section, and conducted a study to obtain qualitative feedback on their viewing experience. While the videos we used include some passive gaze guidance cues, we did not explicitly compare our method to passive (e.g., Nielsen et al. [102]) or active ([82, 57, 108]) gaze guidance techniques, because our method is complementary to them. Our method guarantees viewers see the ROI, whereas passive techniques do not. Active guidance techniques guarantee viewers see a ROI, but they also limit viewer interaction and can reduce immersion, as Nielsen et al. observed, whereas our method does not.

For the study, we used gated clips produced by an earlier version of our system, in which the vertical FOV of each discretized view was equal to the Oculus Rift FOV ($h = 96.02°$), instead of the the full vertical range of $h = 180°$. Using a vertical FOV smaller than the full height might introduce visual artifacts if the viewer looks up or down beyond the view FOV. However, since there was virtually no motion near the poles of these videos, the smaller vertical FOV was not a problem. See Discussion Section for more details on choosing view discretization and FOV.

Each participant watched each of the five videos in one of two conditions: either a Gated version or a Standard (non-looping) version; participants only saw one version of each video. The ordering and condition were random. Each participant saw at least one video in each condition. They watched videos on an Oculus Rift VR headset, while we recorded their head orientation data.

In pilot experiments, in an attempt to single-blind the study, we did not explain the two conditions (Gated and Standard) to the participants. However, we found that because they did not understand the conditions and how they were different, they could not specify which version they preferred. Thus, in order to capture viewers' preferences between Standard and Gated clips, in our main study we informed participants as to which version of each video they were watching.

Before beginning the study, we explained to participants the two versions of videos they might watch; "Standard" version for normal video playback, and "View-Dependent" (Gated) version in which playback would wait for them if they were looking in the wrong direction when an important story element occurs. Before showing each video, we only told participants whether the video was "Standard" or "View-Dependent."

After each video, participants removed the headset and took a break while filling out a questionnaire. We asked the participants to describe the story in their own words and to share feedback on how natural they thought the video playback was.

After all 5 videos were shown, we asked participants to complete a survey comparing the Standard and Gated versions on three 7-point Likert items: how easy it was to follow the stories, how stressful it was to follow the stories, and how interested they were in the stories. At the end, the survey included a binary-choice question asking which version they preferred overall and also included free-response questions asking what they liked and disliked about each version.

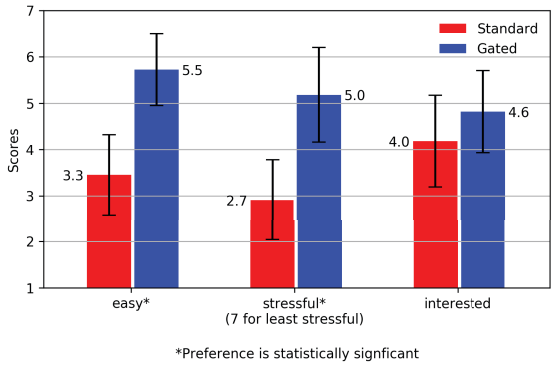


Figure 4.12: User study scores comparing Standard and Gated clips in three categories: (1) how easy it was to follow the story, (2) how stressful it was to follow the story (7 for least stressful), and (3) how interested they were in the stories. Confidence intervals are computed as 2×Standard Error. We found significant differences between the scores for "how easy" and "how stressfull", as indicated by the *'s, but not for "how interested".

There were a total of 11 (5 female and 6 male) participants, with ages ranging from 24 to 36. All participants had some level of prior VR experience, such as watching VR videos or playing VR games. All except one participant watched all 5 videos; one participant preferred not to watch a VR horror video (Stranger Things) but watched the other 4 videos. We instructed participants to stop if they felt sick, but no participants reported sickness during the study.

4.6.1 Study Results

Overall Preference: Standard vs Gated Clips. Most participants (9 out of 11) preferred the Gated videos over the Standard videos. Only one participant preferred the Standard version, because they thought the Gated videos shown were slow (however, their complaints were largely about the pacing in the Gated version of the Lions video). The other participant was ambivalent: they preferred the Standard version if there were strong guidance cues on where to look, but if there were no strong cues, and if the event triggered immediately after they looked at the right thing, they preferred the Gated version.

How Easy, Stressful, Interesting? Participants scored the Standard and Gated versions in three 7-point Likert items: how easy it was to follow the stories, how stressful it was to follow the stories, and how interested they were in the stories (Figure 4.12). For each category, we performed the Wilcoxon signed-rank test on each pair of answers (Standard and Gated) from the same person. We applied continuity correction by adjusting the Wilcoxon rank statistic by 0.5 towards the mean value when computing the z-statistic [70]. For "how easy," there was significant preference for the Gated version ($p < 0.009$, $W = 1.5$, $r = 0.02$). For "how stressful," there was a significant preference for Gated version being *less* stressful ($p < 0.03$, $W = 5$, $r = 0.076$). We found no

statistical significance in the "how interested" answers.

Standard vs Gated Clips: Likes and Dislikes. Most participants liked the Standard video because it did not hold up the story and had better flow, but they did not like the fact that they had to worry about missing important narrative elements. They liked the Gated video for being able to explore scenes at their own pace without worrying they might miss something. One stated reason for disliking the Gated version was having to look around and figure out what to look at to trigger the next event. Some participants disliked the fact that sometimes looking at the right thing did not immediately trigger the next story event. This happened occasionally when the clip they watched did not have seamless forward jump arcs to take them to right before the gate time; in such cases, they had to wait until the video played normally to the gate timecode.

How Natural was the Playback? Most participants interpreted the question "how natural did the playback seem to you" broadly, answering in terms of how natural the story content was, how natural it felt to have the interactive component in the story, or how natural the size of characters in the stories appeared to them (e.g., the rabbit from Invasion! was larger than real-life rabbits). Some participants were not accustomed to live-action videos waiting for them (e.g., the character breathing and waiting), and so thought the interactive aspect was unnatural. Only one participant noticed a ghosting artifact of two distant, moving pedestrians in the Murder Mystery video. In the original video, the two pedestrians walk steadily away from the camera the entire clip (which does not provide a view-dependent video texture an opportunity to loop), and in the postprocess stage, our tool could not find good backward arcs that end before the earliest cut t_C.

ROI Hit Rate for Standard Clips. In addition to the qualitative feedback, we also analyzed the head orientation data of all participants. In Gated versions of videos, participants had to see the ROI at the corresponding gate timecodes in order to proceed. We checked how often participants who watched the Standard version missed the ROI at the same gate timecodes. Overall, only an average of 61.9% of participants saw the ROI at the corresponding times ($\sigma = 31.4\%$).

Time Elapsed for Gated Clips. We also looked at how long it took participants to pass a gate, relative to the length of the gated clip without any looping. For Gated clips in which forward arcs were not enabled, we found that participants took on average 2.25 times the original clip length to pass the gate ($\sigma = 1.77$, average delay of 7.9s). For Gated clips in which forward arcs were enabled, participants took 1.79 times the clip length to pass the gate ($\sigma = 1.15$, average delay of 4.7s). However, this varied considerably; for example, in the second gate of Murder Mystery, viewers passed the gate faster with forward arcs than with the Standard version.

4.7 Discussion

User Study Conclusions. In our user study, most viewers reported they preferred Gated videos over Standard ones. Overall, they find it easier and less stressful to follow stories in Gated than in

Standard videos. However, because participants were aware of which videos were produced by our system, their feedback may be biased. The study indicated that a disadvantage for Gated videos was the need to figure out where to look in order to pass a gate. Thus, we suggest filmmakers use gaze guidance techniques in conjunction with our gating method, such as motion or lighting cues [55], in order to direct viewers' attention.

Design Choices for Gated Video. In order to produce video textures that create a good experience, we recommend directors take the looping structure into account when creating gated clips. In particular, directors should pay attention to structured (i.e., non-periodic, non-stochastic) motion within the scene. Views that do *not* satisfy the gate condition need to be looped, so the director should design the shot, e.g., shoot for a longer period, so that those views have some period of time with no structured motion. For example, video of a car moving across one view cannot be seamlessly looped, because no two frames have the car in the same position. However, if the director films for a longer period of time and captures additional footage of the car moving out of the view, or of the car coming to a stop within the view, then our method could find seamless loops using just the frames after the car leaves, or of the car at rest.

Our method may loop structured motion spanning multiple views. For example, consider a car moving from left to right across most of the scene. In a middle view between the starting and ending views of the car our system might generate a backward arc transitioning from a frame after the car leaves the view to a frame before the car enters the view. A viewer looking at this middle view might then see the car pass through the scene repeatedly. If the car is in the background, it may be fine for viewers to see the car loop in this manner. However, if the car is an important object that the director wants viewers to see, then seeing the car repeat its motion could be confusing. Thus, the director should choose the gate time and ROI carefully in such cases. For example, in the Patio video from our supplemental material, a character stands on the right side of the scene, walks to the left side, and stops. If the director places the ROI on the character after she comes to a stop, the graph-cut algorithm produces arcs that loop the walk, which may not be desirable. If the director instead places the gate ROI on the character *before* she starts walking, as shown in our supplemental material, the director can prevent viewers from seeing the walk loop repeatedly.

Number of View Discretizations & View FOV. The director should consider the trade-offs when choosing parameters for the number of view discretizations $|\mathbb{V}|$ and the FOV per view $v_{i=1...|\mathbb{V}|}$. Recall that, during playback, our video player looks up the nearest discretized v_i and follows the arcs in v_i. Thus, increasing $|\mathbb{V}|$ increases the chance that, at run-time, the actual viewer's FOV will completely overlap with a view v_i, and thereby reduces the chance of seeing artifacts when following arcs in v_i. However, a larger number of views also increases computational cost.

Arc computations for each view only consider the pixels within the view FOV. Thus, a view FOV that is smaller than the FOV of the head-mounted display (HMD) may introduce artifacts during playback, since the HMD would show pixels that fall outside of the corresponding view FOV

when playing loops. A view FOV that is larger than the HMD FOV makes arc computations more conservative and reduces the flexibility in finding seamless arcs, since it includes costs of pixels that fall outside the HMD FOV which viewers actually see. In our examples, we used $|\mathbb{V}| = 40$ and a view FOV of ($w = 80.65°$, $h = 180°$), which we found to be a good trade-off.

4.8 Limitations and Future Work

Our algorithm does not account for audio when generating the view-dependent video textures. By default, our tool simply cross-dissolves the audio during transitions; we have found that this approach usually hides the seams in the loops well for ambient or environmental audio. For clips that have structured audio, directors may need to handle the audio tracks separately when generating gated clips. For example, music tracks could be looped independent of the video [119], whereas audio cues must be carefully synced. Future work could explore ways of looping audio in conjunction with the video.

While our algorithm can loop structured motion spanning multiple views, it may not be able to find seamless loops for *intra-view* structured motion, i.e., structured motion contained within one view. Future work could improve the applicability of our approach, by combining view-dependent arcs with motion segmentation and/or using frame synthesis for better looping video generation. For instance, if a view contains two people performing different repetitive actions, our method may not be able to find a seamless loop, but segmentation approaches [120, 59] could segment the two people and loop them separately. Frame synthesis could generate new frames to increase looping flexibility within a view.

Gated clips open up a considerable design space for the filmmaker to work within when creating their desired experience. For example, should the viewer be required to dwell on the ROI for the gate to be passed? Should it be sufficient that the viewer has seen the ROI at some time in the past? In theory, a gate could be used for every single important moment in the story, but such an arrangement might introduce awkward pauses and disrupt the pacing of the story, so there is also a space for determining where and how to place gates in a narrative. Future work could examine how different types of gate conditions and combinations thereof help achieve a variety of narrative goals.

4.9 Chapter Summary

In this chapter, we focused on the task of manipulating time in 360° video and identified a set of task-dependent realism constraints for this task. We showed that the important properties to maintain (M) are within-view spatiotemporal coherency. We also showed that we can relax (R) linear time and out-of-view spatiotemporal coherency to consider a larger space of visual content and feasible transitions. Specifically, inspired by video textures, we demonstrated that we can play

frames out-of-order to indefinitely extend the video length while waiting for the gate condition to be met. However, unlike conventional video textures, we can relax the criteria for finding frame transitions by only requiring the transition to be imperceptible within the viewer's field of view during playback. The input parameters (I) for this task are the gate timecode, a region of interest (ROI), and a viewing condition to pass the gate.

We then presented a new video player for authoring gated clips. Our tool applies the task-dependent realism constraints M and R as the user manipulates I and produces realistic-looking gated clips. We demonstrate use cases of our tool for various narrative goals, e.g. making sure viewers see important story elements, or making sure viewers do *not* see something that is intended to occur offscreen. We believe that gated clips enhance the ways in which filmmakers tell stories and improve the viewing experience of 360° videos.

Chapter 5

Conclusion and Future Work

This thesis explored task-dependent realism with two manipulation tasks: 1) adjusting object size and position in photographs, and 2) enforcing gated clips for 360° video. For each task, given user-specified input parameters (I), this thesis identified explicit spatial and temporal constraints to relax (R) and maintain (M), such that manipulating I would generate realistic-looking outputs. While we focused on two specific tasks, this thesis shows the advantage of task-dependent realism at a broader level – by explicitly identifying and applying realism constraints for a given task, we can significantly increase the space of realistic-looking solutions and design tools which automatically preserve important realism properties as the user explores edits.

A critical theme in this thesis is to develop a deep understanding of what assumptions and constraints can be *relaxed* for a given manipulation task. Often times, these constraints are counter-intuitive and challenge our assumptions about the world (e.g., curved light rays can produce plausibly realistic images). However, by concentrating on a single task instead of multiple tasks, we are able to deep dive into the problem and relax constraints to the maximum extent while ensuring the outputs remain realistic. By doing so, we significantly increase the space of candidate solutions, which then makes it easier to design an underlying representation for task manipulation. For example, in ZoomShop, we devised a non-linear camera model to parameterize the space of edits. For gated clips, we used a graph formulation to find a set of backward arcs to enforce gates. The advantage of task-dependent realism lies in the increased space of feasible solutions, new representations to explore the solution space, and the convenience of automatically preserving realism while editing.

We speculate that task-dependent realism offers an enhanced way to to edit visual media – by applying task-dependent realism constraints, we free users from the burden of maintaining realism and allow them to better focus on their authoring goals. Here, we identify a few directions of future work.

5.1 Identifying Constraints

In this thesis, we manually identified explicit constraints to relax and maintain for specific tasks. However, manually finding these constraints can be time-consuming; it requires a deep understanding of perceptual changes that the task creates and often involves trial-and-error. One direction of future work is to systematically derive these constraints for a given task.

Extracting Constraints Future work could find general realism constraints from learned machine learning models and use them as a starting point for curating task-specific realism constraints. Many AI models, such as generative models, are trained on large amounts of real image data and have implicitly learned what makes a photo real or fake. However, these"rules" for constructing a realistic image are encoded as millions of model parameters and are not explicitly defined. Future work could investigate ways to extract explicit realism constraints from these learned models and refine them for specific manipulation tasks.

Refining Constraints To refine realism constraints for a given task, one approach is to randomly select constraints to maintain and relax and check whether the result looks realistic. Future work can explore various strategies for finding these constraints. For example, one can start off with many constraints to maintain and test out various constraints to relax (R), or start off with very few constraints to maintain (M) and progressively add more as necessary, or some combination of both.

Evaluating Realism When curating task-dependent realism constraints, it is important to check whether applying the selected constraints would lead to realistic results. Although we manually inspected results in this work, future work can use data-driven techniques to speed up the process. For example, one could use AI models to evaluate results by comparing them with the learned distribution of realistic images. Alternatively, one could use crowd-sourcing to streamline the evaluation process (e.g., Amazon Mechanical Turk).

Applying Constraints to Other Tasks In this work, we focused on one single task at a time and identified task-specific realism constraints for each task. Another direction for future work is to reverse the process; given task-dependent realism constraints for one task, what other tasks might these constraints be useful for? For example, in ZoomShop, one of the relaxed constraints is that light rays do not need to be straight to produce realistic images. It would be interesting to explore how this insight may apply to other image editing tasks, such as re-lighting or re-coloring.

5.2 Exploration of Edits

In this work, based on identified task-dependent realism constraints, we designed representations of solution spaces such that it was easy to optimize for the specific tasks. For example, in ZoomShop, we used a non-linear camera model to parameterize the 3D scene and smoothly vary changes in scale across depth. For gated clips, we used a graph representation for ease of finding good backward transitions. In these projects, while we used optimization techniques to find good results, we did not explicitly design the representations such that the solutions changed in a predictable way as the input parameters changed (e.g., direct manipulation). For example, when specifying parameters for gated clips, there is no guarantee that tweaking the gate parameters would lead to a similar set of backward arcs and transitions; the resulting arcs could be completely different.

Future work could investigate ways to design more intuitive representations or manifolds of the solution space or better interfaces for navigating the solution space when exploring edits. One option is direct manipulation interfaces, where there is a continuous representation between the input and the output solution. Other possible enhancements include multi-modal interfaces (e.g., voice, text, and haptic), which would support multiple ways of exploring the solution space.

In this work, we gave users the freedom to specify input parameters and explore edits at their own pace. Future work could examine ways to provide additional guidance during the exploration process. For example, future work could give real-time suggestions and feedback to users to help them converge on a satisfactory solution more quickly. One approach is to use data-driven techniques to learn about the user's preferences based on their exploration history and make recommendations derived from those insights. Other possibilities include crowd-sourcing ideas, suggesting templates, and showing similar artistic examples.

5.3 Generating and Editing Synthetic Content

While this thesis focused on manipulation of real visual media, future work could explore ways to extend task-dependent realism to the generation and manipulation of synthetic content. Today, generative models [53, 63] are popular tools for creating synthetic images. While powerful, these generative models tend to be more difficult to control than non-AI techniques. It would be interesting to explore how task-dependent realism may help further enhance the capabilities of these models.

Generating Synthetic Content Currently, there are two main challenges in generating synthetic content with AI models. One of them is providing sufficient user control (i.e., generating an output which the user wants). Many models nowadays rely on prompt-based (text-based) interfaces, which has a lot of room for interpretation. Given that models must then map these lower-dimensional prompts to high-dimensional images, there is even more room for error, making it difficult for users to get the output they want. The second challenge is producing realistic outputs; generative models

are typically good at producing content that is similar to the distribution of data that they were trained on. However, if the user prompts the model to generate an image that is out-of-distribution of its training data, the model may generate a very unrealistic output (e.g., a cat with two heads instead of two individual cats). As such, future work could explore how task-dependent realism may play a role. By focusing on one specific task and identifying explicit realism constraints for that task, one may be able to better train the models with those insights and input controls. This may lead to more realistic results for that task as well as greater user control for that task.

Editing Synthetic Content After generating synthetic content, users may wish to edit some aspects of it. The main challenge here is to perform the manipulation while preserving structure and realism. Because of the under-constrained problem of mapping lower-dimensional input (e.g., text) to high-dimensional images, changing the input by a small amount may lead to very different results. Many works [61, 29, 104, 106, 52] have taken steps towards this goal by manipulating the latent representation of images. Again, future work could integrate task-dependent realism constraints into these models to enhance their performance on specific tasks. Equipped with task-dependent realism, one may be able to design task-specific models with greater user control that maintains structure and realism as the user performs edits.

5.4 Visual Perception

The task-dependent realism constraints in this thesis reveal that there is quite an amount of "wiggle room" for human visual perception; we can relax many constraints and assumptions and still produce realistic results. One direction of future work is to use visual perception principles to systematically guide the derivation of task-specific constraints. For example, instead of trial-and-error, perception literature may offer guidance on an efficient strategy for selecting and testing out task-dependent realism constraints.

Conversely, it would be interesting to explore how task-dependent realism could inform principles in vision and perceptual science. For example, while artists and vision scientists have noticed that linear perspective does not accurately capture human visual perception of a real scene, there is no consensus on a model of visual perception. Perception scientists could perhaps use task-dependent realism constraints from ZoomShop or use ZoomShop as an experimental tool for developing a more comprehensive model of visual perception.

5.5 Non-Visual Domains

While this thesis focused on manipulating visual media, future work could extend task-dependent realism to non-visual domains as well, such as audio or haptic media. Some works explore the

perceptual limits of different domains, such as haptic illusions in virtual reality [1]. Future work can apply these insights in determining realism constraints for specific tasks. In addition, future work can investigate tasks that involve more than one type of domain (e.g., visual and non-visual) and explore strategies for finding realism constraints with the additional dimensions.

Appendix A

ZoomShop

A.1 Geometric Description and Derivation of $b(z)$

Here, we show how our $b(z)$ camera parameterization can be understood geometrically in terms of non-linear camera models. We first show that a piecewise-linear, continuous choice of $b(z)$ corresponds to a sequence of linear camera models, each applied to different depth ranges, equivalent to the Computational Zoom model proposed by Badki et al. [14]. We then describe generalizations to non-linear and non-continuous $b(z)$, and what these correspond to geometrically. For each type of these $b(z)$ (i.e., piecewise linear, curved, and discontinuous), we show that Equation A.1 holds:

$$u = \frac{x}{b(z)} \tag{A.1}$$

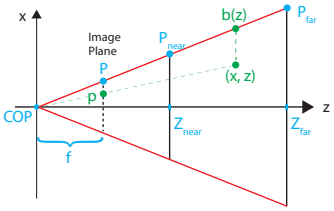
Figure A.1 shows the boundary curve $b(z)$ for linear perspective, piecewise linear, and curved camera models.

A.1.1 Piecewise Linear Camera Model

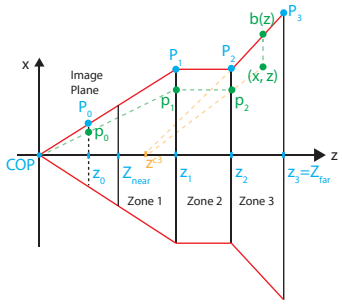
We first explore a piecewise linear camera model. In this model, separate linear cameras are applied to each depth range, with continuity constraints between the cameras. We show how the $b(z)$ formulation can be derived from this model. Note that this model is equivalent to Computational Zoom [14], and this shows how Computational Zoom is a special case of our framework.

Figure A.1b shows a piecewise linear camera model. With the camera at the origin, and an image plane at focal depth $f = z_0$, we divide up the scene into a series of *depth zones*, bounded by depth planes $z_1 ... z_N$, where $z_i < z_{i+1}$ for $i \in [0...N-1]$. For each depth plane, the corresponding half-plane width is P_i, which also marks the view boundary at z_i.

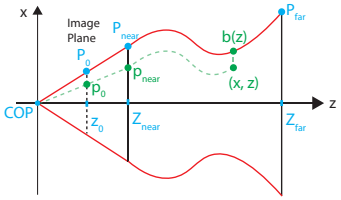
Virtual Camera Positions. In the piecewise linear camera model, we can treat each individual



(a) Linear Perspective Camera Model. P is the half-width of the image plane in world space. Z_{near} and Z_{far} are the z-values of the near and far plane, and f is the focal length of the camera.



(b) Piecewise Linear Perspective Camera Model. In this example, the number of depth zones $N = 3$. A scene point (x, z) gets iteratively projected onto planes P_2, P_1, and then P_0 to form the final image. Each depth zone i obeys linear perspective, with P_{i-1} as the virtual image plane and z^{c_i} as the virtual camera. z^{c3} is shown in yellow.



(c) Curved Camera Model. Parameterized by a boundary curve $b(z)$ for $z \in [Z_{near}, Z_{far}]$. A scene point (x, z) is projected onto the near plane $P_{near} = b(Z_{near})$ at $(p_{\text{near}}, Z_{\text{near}})$, and then onto the image plane at (p_0, z_0).

(d) Comparison of using different camera models, in a 3D scene inspired by Burleigh et al. [27]. In this example, our aim is to scale up the bust while keeping the couches visible. Using linear perspective, the only way is to zoom in crop, which cuts out part of the couch. Using the piecewise linear camera model, we divide the scene into three zones (yellow, green, non-colored) and scale each zone separately. However, this introduces a seam at the zone boundary between the non-colored and green zone (see red circles). Using curved camera rays, we achieve a smoother transition of scale between zones and removes the seams (see green circles). In the last two images, pixels that are not in the original photo are colored in cyan.

Figure A.1: Camera Models

depth zone i as a conventional linear perspective projection, with P_{i-1} as the "image plane" of that zone, and z^{c_i} as the position of the virtual camera. See Figure A.1b for an illustration.

For each zone i, we can compute the virtual camera position z^{c_i} by extrapolating where the bounds of zone i will hit the z-axis. Take the example in Figure A.1b. The line connecting (z_2, P_2) and (z_3, P_3) is defined by:

$$l(z) = P_2 + \frac{P_3 - P_2}{z_3 - z_2}(z - z_2) \tag{A.2}$$

The line will intersect the z-axis at z^{c_3}:

$$l(z^{c_3}) = P_2 + \frac{P_3 - P_2}{z_3 - z_2}(z^{c_3} - z_2) \Longrightarrow 0 \tag{A.3}$$

$$z^{c_3} = z_2 - \frac{P_2(z_3 - z_2)}{P_3 - P_2} \tag{A.4}$$

In general, for any depth zone i, we can compute z^{c_i}:

$$z^{c_i} = z_{i-1} - \frac{P_{i-1}(z_i - z_{i-1})}{P_i - P_{i-1}} \tag{A.5}$$

To compute the final position of a scene point (x, z) on P_0, we iteratively project (x, z) onto intermediate "image planes" P_{i-1}, P_{i-2}, ..., until the final image plane P_0.

Projection in Closed Form Solution The pseudo-code for projecting a point (x, z) iteratively to P_{i-1}, P_{i-2}... until the image plane P_0 is shown in Figure A.2.

In general, given any point (x, z) in depth zone i, we can figure out its projected point onto the previous plane P_{i-1} via $p_x = \frac{f}{z}x$. Offsetting f and z to be with respect to the virtual camera z^{c_i}, the projected position (x', z') onto the previous plane P_{i-1} is:

$$x' = \frac{z_{i-1} - z^{c_i}}{z - z^{c_i}}x \tag{A.6}$$

$$z' = z_{i-1} \tag{A.7}$$

We can convert the pseudo-code iterative projection to closed form. First, we substitute z^{c_i} in Equation A.6 (also line 9 of the pseudo-code) with Equation A.5:

$$x' = \frac{z_{i-1} - z^{c_i}}{z - z^{c_i}} x \tag{A.8}$$

$$= \frac{z_{i-1} - (z_{i-1} - \frac{P_{i-1}(z_i - z_{i-1})}{P_i - P_{i-1}})}{z - (z_{i-1} - \frac{P_{i-1}(z_i - z_{i-1})}{P_i - P_{i-1}})} x \tag{A.9}$$

$$= \frac{\frac{P_{i-1}(z_i - z_{i-1})}{P_i - P_{i-1}}}{z - z_{i-1} + \frac{P_{i-1}(z_i - z_{i-1})}{P_i - P_{i-1}}} x \tag{A.10}$$

$$= \frac{P_{i-1}(z_i - z_{i-1})}{(P_i - P_{i-1})(z - z_{i-1}) + P_{i-1}(z_i - z_{i-1})} x \tag{A.11}$$

$$= \frac{P_{i-1}}{\frac{z - z_{i-1}}{z_i - z_{i-1}}(P_i - P_{i-1}) + P_{i-1}} x \tag{A.12}$$

$$= \frac{P_{i-1}}{\frac{z - z_{i-1}}{z_i - z_{i-1}} P_i + (1 - \frac{z - z_{i-1}}{z_i - z_{i-1}}) P_{i-1}} x \tag{A.13}$$

$$= \frac{P_{i-1}}{\frac{z - z_{i-1}}{z_i - z_{i-1}} P_i + \frac{z_i - z}{z_i - z_{i-1}} P_{i-1}} x \tag{A.14}$$

The denominator $\frac{z - z_{i-1}}{z_i - z_{i-1}} P_i + \frac{z_i - z}{z_i - z_{i-1}} P_{i-1}$ is just a linear interpolation between the zone boundary endpoints at P_{i-1} and P_i. In fact, the denominator is equal to the boundary point $b(z)$ at (x, z).

In other words, we can rewrite Equation A.14 and A.6 as:

$$x' = \frac{P_{i-1}}{b(z)} x \tag{A.15}$$

(x', z') is the projection of (x, z) to plane P_{i-1}. Following the pseudo-code, in the next iteration, we project (x', z') to P_{i-2}. Let the projected point onto P_{i-2} be (x'', z''). Then, using Equation A.14:

$$x'' = \frac{P_{i-2}}{b(z')} x' \tag{A.16}$$

$$= \frac{P_{i-2}}{b(z_{i-1})} x' \tag{A.17}$$

$$= \frac{P_{i-2}}{P_{i-1}} x' \tag{A.18}$$

$$z'' = z_{i-2} \tag{A.19}$$

where $b(z_{i-1}) = P_{i-1}$ because (x', z') is the projected point on plane P_{i-1}, and the boundary at $z' = z_{i-1}$ is precisely P_{i-1}.

By the same logic, in subsequent iterations, when projecting (x_j, z_j) at plane P_j onto P_{j-1}, the

```
# i is zone index of point (x,z).
project(x, z, i):
    # If in zone 1, just project onto P_0
    if i == 1:
        return p_0 = (z_0 / z) x
    else:
        # Compute the projected point (x',z') onto P_{i-1}
        # z^{c_i} is position of zone i's 'ghost' camera
        x' = ((z_{i-1} - z^{c_i}) / (z - z^{c_i})) x  # See Equations 8 and 9
        z' = z_{i-1}
        return project(x', z', i-1)
```

Figure A.2: Piecewise Linear: Pseudo-code for projecting a point (x, z) iteratively onto the image plane P_0.

scale factor for x_j is $\frac{P_{j-1}}{b(z_j)} = \frac{P_{j-1}}{P_j}$. Thus, the closed form solution for projecting (x, z) to the image plane P_0 is:

$$p_0 = \frac{P_0}{P_1}\frac{P_1}{P_2}\ldots\frac{P_{i-2}}{P_{i-1}}\frac{P_{i-1}}{b(z)}x \tag{A.20}$$

$$= \frac{P_0}{b(z)}x \tag{A.21}$$

where p_0 is the projected (world) coordinate of x onto image plane at z_0.

The normalized image coordinates $u_0 \in [-1, 1]$ of the projection is:

$$u_0 = \frac{1}{P_0}\frac{P_0}{b(z)}x \tag{A.22}$$

$$= \frac{x}{b(z)} \tag{A.23}$$

where P_0 is the half-width of the image plane in world space.

Another way to interpret this piecewise linear model is in terms of light paths. In a conventional pinhole camera model, light follows a straight line from a scene point to the camera's focal center and intersects the image plane. In the piecewise linear model, a light path from a scene point follows a sequence of straight lines, bending at each depth zone boundary (Figure A.1b).

This geometric interpretation of light paths can be extended to the other camera models as well. For all camera models in our framework, the light paths follow the shape of $b(z)$, i.e., for a scene point at position (x_0, y_0, z_0), the light path is a curve given by $f(z) = (\frac{x_0}{b(z_0)}b(z), \frac{y_0}{\lambda b(z_0)}b(z), z)$, where $\lambda = H/W$ is the image aspect ratio.

A.1.2 Curved Paths

We can generalize the piecewise linear model to a curved model (Figure A.1c). In this generalized form, camera paths are no longer piecewise linear, but general curves: each light path is a scaled version of $b(z)$.

Equation A.1 still applies to the general view boundary curve. Instead of a finite number of piecewise linear depth zones defined by a piecewise linear $b(z)$, we now have a curved $b(z)$, which is equivalent to an infinite number of piecewise linear depth slices, where each slice is infinitesimally thin.

In this curved model, the nearest depth plane (before the image plane) of the scene is $P_{\text{near}} = b(Z_{\text{near}})$, as shown in Figure A.1c. Given any scene point (x, z), the projected point onto P_{near}, is:

$$p_{\text{near}} = \frac{P_{\text{near}}}{b(z)} x \tag{A.24}$$

Next, the projection from $(p_{\text{near}}, Z_{\text{near}})$ onto the image plane P_0 is just conventional linear perspective:

$$\begin{aligned} p_0 &= \frac{f}{z_{\text{near}}} p_{\text{near}} && \text{(A.25)} \\ &= \frac{z_0}{z_{\text{near}}} p_{\text{near}} && \text{(A.26)} \\ &= \frac{z_0}{z_{\text{near}}} \frac{P_{\text{near}}}{b(z)} x && \text{(A.27)} \\ &= (z_0 \frac{P_{\text{near}}}{z_{\text{near}}}) \frac{x}{b(z)} && \text{(A.28)} \\ &= P_0 \frac{x}{b(z)} && \text{(A.29)} \end{aligned}$$

where P_0 is the half-width of the image plane in world space. We can convert p_0 from world coordinates to normalized image coordinates $u_x \in [-1, 1]$:

$$\begin{aligned} u_x &= \frac{p_0}{P_0} && \text{(A.30)} \\ &= \frac{x}{b(z)} && \text{(A.31)} \end{aligned}$$

A.1.3 Discontinuous Paths

We can further generalize the continuous model to a discontinuous one. Figure A.3 shows a piecewise discontinuous model, where camera paths follow discontinuous lines. We show that Equation A.1

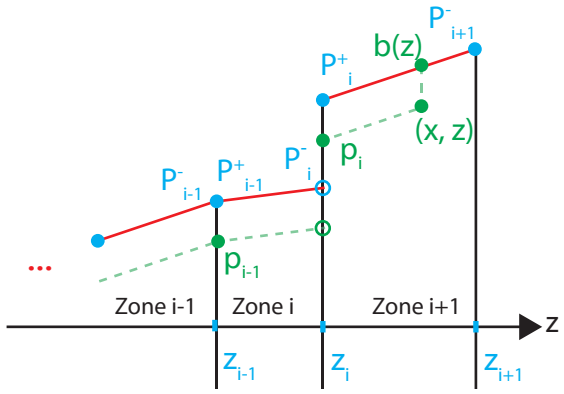


Figure A.3: Illustration of discontinuous piecewise-linear camera paths. We break off each half-width depth plane P_i into two parts, P_i^- and P_i^+. In this example, a discontinuity occurs at z_i because $P_i^- \neq P_i^+$. $b(z)$ is continuous at z_{i-1} because $P_{i-1}^- = P_{i-1}^+$.

holds even if $b(z)$ is discontinuous.

For piecewise discontinuous paths, we can define two half-widths P_i^+, P_i^- for each depth plane:

$$b(z) = \frac{z - z_{i-1}}{z_i - z_{i-1}} P_i^+ + \frac{z_i - z}{z_i - z_{i-1}} P_{i-1}^-, \quad \text{where } z_{i-1} \leq z < z_i \tag{A.32}$$

When $P_i^+ \neq P_i^-$, a discontinuity occurs in the camera paths through this point. See Figure A.3 for an illustration. We can imagine an infinitesimally thin depth zone at each $z_i, 1 \leq i \leq N$, so:

$$p_0 = \frac{1}{P_0^+} \cdot \cdots \cdot \frac{P_{i-1}^+}{P_i^-} \frac{P_i^-}{P_i^+} \frac{P_i^+}{b(z)} x \tag{A.33}$$

which converts to Equation A.1 in image coordinates.

A.2 Removing Artifacts

Scaling depth ranges can lead to disocclusions, pixel stretching, or shearing. For example, the teaser figure shows both disoccluded regions (behind the left tree branch) and sheared pixels on the lake in cyan between the yellow and green depth zones. Both of these issues introduce artifacts, which we address with two heuristics.

In the first heuristic, we check the amount of shearing of each pixel in the final image. If a pixel is significantly sheared, it most likely saddles between two different depths that were scaled differently. This may lead to visible artifacts and/or disocclusion. To check for shearing, in the fragment shader, we check the dot product between vectors $\frac{d\vec{u}}{dx} = \begin{bmatrix} \frac{du}{dx} & \frac{dv}{dx} \end{bmatrix}^T$ and $\frac{d\vec{u}}{dy} = \begin{bmatrix} \frac{du}{dy} & \frac{dv}{dy} \end{bmatrix}^T$. If the vectors are orthogonal, then there's no shear. But if the dot product is greater than some threshold τ_{shear}, then

we make the pixel transparent:

$$\frac{d\vec{u}}{dx} \cdot \frac{d\vec{u}}{dy} > \tau_{\text{shear}} \tag{A.34}$$

We use $\tau_{\text{shear}} = 0.53 - 0.9$ in our results.

In the second heuristic, we check for the amount of non-uniform scaling (stretching). A pixel that's stretched significantly in the x-direction, again, likely lies between two different depths that were scaled differently, and thus introduces disocclusion or artifacts. To check for non-uniform scaling, in the fragment shader, we compute the ratio of $||\frac{d\vec{u}}{dx}||$ and $||\frac{d\vec{u}}{dy}||$. A ratio of 1 means uniform scaling; any deviation means the scaling is non-uniform. If the ratio is less than some threshold $\tau_{\text{nonuniform}}$, then we make the pixel transparent:

$$\frac{||\frac{d\vec{u}}{dx}||}{||\frac{d\vec{u}}{dy}||} < \tau_{\text{nonuniform}} \tag{A.35}$$

We used $\tau_{\text{nonuniform}} = 0.2 - 0.3$ in our results, and also apply the same check for the y-direction, i.e., $||\frac{d\vec{u}}{dy}||/||\frac{d\vec{u}}{dx}||$.

A.3 Additional Translation Results

In Figure A.4, scaling up the birds pushed the left rock out of view. In addition to translation constraints on the rock, we added an additional constraint on the float to keep it fixed in place. The output shows the rock translated along with some connected water in front. In Figure A.5, compressing the depth in the valley pushed the two side rocks partially out of view. Under the shown constraints, ZoomShop smoothly translates the rocks towards the center as well some ground in front.

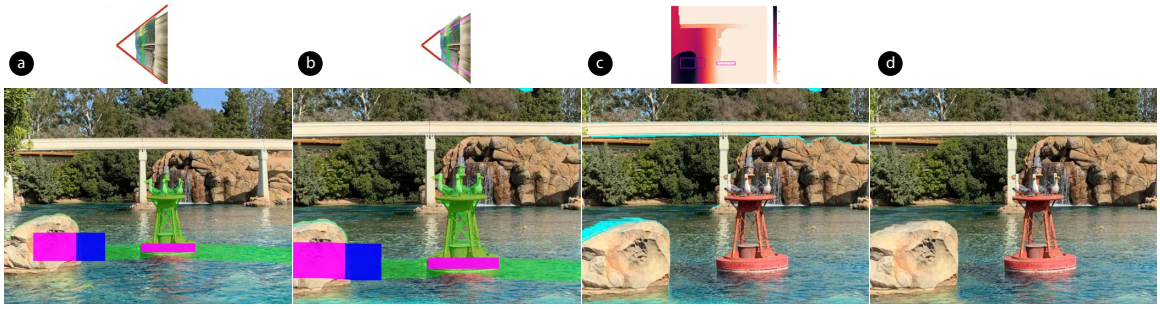


Figure A.4: Birds. Goal: Scale up birds while also keeping the left side rock visible. Magenta and blue rectangles are source-destination pairs which are input to our translation optimization. Each pair of rectangles has the same size; magenta overlays blue rectangles. (a) Original photo. (b) Scaled up birds. (c) ZoomShop output (top: translation map). (d) ZoomShop with inpainting (automatic).

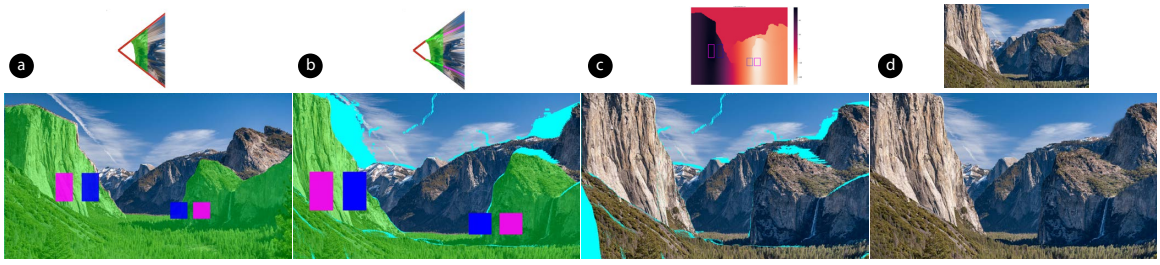


Figure A.5: Yosemite [141]. Goal: Compress depth in valley while keeping two side rocks in view. Magenta and blue rectangles are source-destination pairs which are input to our translation optimization. (a) Original photo. (b) Compressed valley. (c) ZoomShop output (top: translation map). (d) ZoomShop with inpainting (top: automatic, bottom: manual guidance)

Appendix B

Gated Clips

B.1 View-Dependent Arc Cost Computation

We now provide more detail into how arc costs are computed. Recall that an arc is a transition between two frames (t, t'), and a view-dependent arc is a transition between two frames in a particular view v, represented as a triplet (v, t, t').

Our goal is to find arcs that transition seamlessly, so that viewers don't notice the transitions when they occur. In a pre-processing step, we assign costs to all possible view-dependent arcs (v, t, t') within a gated clip. The cost measures how seamless the arc transition is. As mentioned before, we only consider the field of view that the viewer sees when computing view-dependent arcs. We first discretize the view-sphere of all possible viewing directions into V views, and for each view, compute the costs of all possible arcs within that view. Therefore, the total number of arcs is $V * f^2$, where f is the number of frames in the gated clip.

B.1.1 View-Dependent Arc Cost Matrix

There are f^2 total arcs in each viewing direction, where f is the number of frames in the gated clip. We construct a cost matrix of size $f \times f$ that represents the arc cost between each pair of frames.

We define the cost of an arc between two frames for a given view as follows. Our goal is to penalize visually-noticeable changes when cross-dissolving between the two frames, such as a person appearing or disappearing, while ignoring minor changes due to pixel noise. We then apply a user-defined threshold to determine if an arc is noticeable or not.

The cost for an arc from time i to time j in view direction k is a summation over every pixel visible to the view, comparing the frames before and after the arc, summed over the duration N of

the cross-dissolve:

$$C(v_k, t_i, t_j) = \sum_{x=0}^{N} \sum_{\ell \in \text{pixels}(k)} d(I_{\ell,i+x}, I_{\ell,j+x}) \ \max(e_{\ell,i+x}, e_{\ell,j+x}) \tag{B.1}$$

where $I_{\ell,i}$ is the RGB value of pixel ℓ at time i, and $e_{\ell,i} \in [0,1]$ is a binary edge map at pixel ℓ at time i. The edge maps are computed by Canny edge detection with a 3×3 Sobel filter, and min/max thresholds of 80 and 100 for the intensity gradient. The difference function ignores pixel differences below a threshold τ:

$$d(a,b) = \begin{cases} ||a-b||^2, & ||a-b||^2 \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{B.2}$$

where a and b are 3-dimensional vectors that represent RGB values in the range [0..1]. We use $\tau = 0.015$ to 0.2, depending on how much high-frequency, stochastic motion there is in the clip. The τ threshold prevents pixel changes due to stochastic motion, such as moving tree leaves, from overly penalizing the arc. Empirically, we found that a clip with no trees in the foreground works well with $\tau = 0.015$, whereas a clip with large foreground trees moving in the wind requires a larger $\tau = 0.2$.

For computational efficiency, we scaled our 360° videos in equirectangular format down to 640×320 before computing the cost matrices. We used a Summed Area Table [40] to accelerate computation, since the summation otherwise would include considerable overlapping computations for overlapping views. The Summed Area Table computation for a 7s 30fps clip takes about 3.3 hours to complete on a 3.1 GHz Intel Core i7 processor, in single-threaded unoptimized Python.

B.2 Buffer Edge Costs

We describe in more detail how buffer edge costs are assigned in our graph cut formulation. For views in the direction of the gate ($v \in \mathbb{H}$), we set the buffer edge weights to infinity for all timecodes $t \in 1 : T-1$.

For views $v \notin \mathbb{H}$, by default, we set the weight of the buffer edge from (v,t) to $(v,t)_B$ to the cost of the best backward arc (lowest-cost) from this node, from among all backward arcs that satisfy the user-specified minimum backward arc length. The user specifies a threshold γ for how large a cost is perceptually acceptable. If a node (v,t) has a backward arc with $C(v,t,t') < \gamma$, we call the node a *safe* node; otherwise, we call it an *unsafe* node. With our view discretization and FOV, it is possible that the viewer's FOV may partially extend outside the FOV used to compute an arc cost, so even though $C(v,t,t')$ may be less than γ, $C(v+\epsilon,t,t')$ may not be and may have ghosting in the periphery. Hence, we prefer taking arcs from nodes that are neighbored by safe nodes, in order to decrease the likelihood of peripheral ghosting.

Thus, to compute the buffer edge weight $E_B(v,t)$, we find contiguous blocks of safe nodes in each viewing direction, i.e., each row of the graph (Figure 4.8), and add a small penalty to arcs of safe nodes that are within K frames near the ends of the blocks, or do not fully overlap with any blocks in adjacent views:

$$E_B(v,t) = \begin{cases} \infty, & \text{if } v \in \mathbb{H} \\ \omega_1(v,t) + \omega_2(v,t), & \text{if } v \notin \mathbb{H} \text{ and } (v,t) \text{ is safe} \\ \min\limits_{t' \leq t-M} C(v,t,t'), & \text{if } v \notin \mathbb{H} \text{ and } (v,t) \text{ is unsafe} \end{cases} \tag{B.3}$$

where M is the minimum loop length in number of frames. ω_1 assigns a small penalty (between 0 and 2) based on how close the node is to the ends of the contiguous block as well as the block length, and ω_2 assigns a small penalty based on whether the block it is in has complete overlap with any block in neighboring views. Let $D(v,t)$ represent the contiguous block of safe nodes that node (v,t) is in.

$$\alpha = \min(t - D_{\text{start}}(v,t),\ D_{\text{end}}(v,t) - t) \tag{B.4}$$

$$\beta = \min\left(K, \frac{D_{length}(v,t)}{2}\right) \tag{B.5}$$

$$\phi = \begin{cases} 1, & \text{if } \beta = \dfrac{D_{length}(v,t)}{2} \\ 0, & \text{otherwise} \end{cases} \tag{B.6}$$

$$\omega_1(v,t) = \begin{cases} 1 - \dfrac{\alpha}{\beta} + \phi, & \alpha \leq \beta \\ 0, & \text{otherwise} \end{cases} \tag{B.7}$$

$$\omega_2(v,t) = \alpha \sum_{v' \in \mathbb{N}(v)} \max_{t'} \delta(D(v,t) \subseteq D(v',t')) \tag{B.8}$$

where δ is an indicator function that shows whether the frames of the first block is a subset of the frames in the second block. α is the amount of penalty for each neighbor in which the block $D(v,t)$ is not a subset of. We used $\alpha = 0.1$ and $K = 15$.

This heuristic reduces the chance of getting bad arcs in the post-process step, because it favors cutting on sequential arcs in the contiguous blocks, as opposed to arcs of non-contiguous, isolated safe nodes. For large contiguous blocks of safe nodes, it is likely that the frames have similar levels of motion (e.g., static), so safe nodes in large contiguous blocks are more likely to have a good backward arc that ends at or before the first safe node of that block.

Bibliography

[1] Parastoo Abtahi and Sean Follmer. Visuo-haptic illusions for improving the perceived performance of shape displays. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Adobe. Adobe photoshop, 2021.

[3] Adobe. Adobe premiere pro, 2021.

[4] Advanced Microcomputer Systems. Dragon's lair. Cinematronics [Arcade version], 1983.

[5] Advanced Microcomputer Systems. Space ace. Cinematronics [Arcade version], 1984.

[6] Aseem Agarwala, Maneesh Agrawala, Michael Cohen, David Salesin, and Richard Szeliski. Photographing long scenes with multi-viewpoint panoramas. In *ACM SIGGRAPH 2006 Papers*, pages 853–861. 2006.

[7] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *ACM SIGGRAPH 2004 Papers*, pages 294–302. 2004.

[8] Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin, and Richard Szeliski. Panoramic video textures. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 821–827, New York, NY, USA, 2005. ACM.

[9] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic multiprojection rendering. In *Eurographics Workshop on Rendering Techniques*, pages 125–136. Springer, 2000.

[10] John M Airey, John H Rohlf, and Frederick P Brooks Jr. Towards image realism with interactive update rates in complex virtual building environments. *ACM SIGGRAPH computer graphics*, 24(2):41–50, 1990.

[11] Yuval Alaluf, Or Patashnik, Zongze Wu, Asif Zamir, Eli Shechtman, Dani Lischinski, and Daniel Cohen-Or. Third time's the charm? image and video editing with stylegan3. In *Computer Vision–ECCV 2022 Workshops: Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part II*, pages 204–220. Springer, 2023.

[12] Moab Arar, Dov Danon, Daniel Cohen-Or, and Ariel Shamir. Image resizing by reconstruction from deep features. *arXiv preprint arXiv:1904.08475*, 2019.

[13] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. SIGGRAPH '07, page 10–es, New York, NY, USA, 2007. Association for Computing Machinery.

[14] Abhishek Badki, Orazio Gallo, Jan Kautz, and Pradeep Sen. Computational zoom: A framework for post-capture image composition. *ACM Trans. Graph.*, 36(4), July 2017.

[15] Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Selectively deanimating video. *ACM Trans. Graph.*, 31(4):66:1–66:10, July 2012.

[16] Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Automatic cinemagraph portraits. pages 17–25, 2013.

[17] Reynold Bailey, Ann McNamara, Nisha Sudarsanam, and Cindy Grimm. Subtle gaze direction. *ACM Trans. Graph.*, 28(4):100:1–100:14, September 2009.

[18] Joseph Baldwin, Alistair Burleigh, and Robert Pepperell. Comparing artistic and geometrical perspective depictions of space in the visual field. *i-Perception*, 5(6):536–547, 2014.

[19] Baobab Studios. Invasion!, 2016. https://www.baobabstudios.com/invasion.

[20] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24, 2009.

[21] Ronen Barzel and Alan H Barr. *Physically-based modeling for computer graphics: a structured approach*. Morgan Kaufmann, 2013.

[22] Kadi Bouatouch and Christian Bouville. *Photorealism in computer graphics*. Springer Science & Business Media, 2013.

[23] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, September 2004.

[24] John Brosz, Sheelagh Carpendale, Faramarz Samavati, Hao Wang, and Alan Dunning. Art and nonlinear projection. 01 2009.

[25] John Brosz, Faramarz F. Samavati, M. Sheelagh T. Carpendale, and Mario Costa Sousa. Single camera flexible projection. In *Proceedings of the 5th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '07, page 33–42, New York, NY, USA, 2007. Association for Computing Machinery.

[26] Gerd Bruder, Frank Steinicke, Phil Wieland, and Markus Lappe. Tuning self-motion perception in virtual reality with visual illusions. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1068–1078, July 2012.

[27] Alistair Burleigh, Robert Pepperell, and Nicole Ruta. Natural perspective: Mapping visual space with art and science. *Vision*, 2(2):21, 2018.

[28] William R. Bussone. Linear and angular head accelerations in daily life. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 7 2005.

[29] Mingdeng Cao, Xintao Wang, Zhongang Qi, Ying Shan, Xiaohu Qie, and Yinqiang Zheng. Masactrl: Tuning-free mutual self-attention control for consistent image synthesis and editing, 2023.

[30] Robert Carroll, Aseem Agarwala, and Maneesh Agrawala. Image warps for artistic perspective manipulation. *ACM Trans. Graph.*, 29(4), July 2010.

[31] Robert Carroll, Maneesh Agrawala, and Aseem Agarwala. Optimizing content-preserving projections for wide-angle images. *ACM Trans. Graph.*, 28(3), July 2009.

[32] Alan Chalmers and Andrej Ferko. Levels of realism: From virtual reality to real virtuality. In *Proceedings of the 24th Spring Conference on Computer Graphics*, pages 19–25, 2008.

[33] Shenchang Eric Chen. Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 29–38, New York, NY, USA, 1995. ACM.

[34] Stephen Chenney and David Forsyth. View-dependent culling of dynamic systems in virtual environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, I3D '97, pages 55–58, New York, NY, USA, 1997. ACM.

[35] Anton Cherepkov, Andrey Voynov, and Artem Babenko. Navigating the gan parameter space for semantic image editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3671–3680, 2021.

[36] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018.

[37] James H Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[38] Patrick Coleman and Karan Singh. Ryan: rendering your animation nonlinearly projected. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 129–156, 2004.

[39] J. P. Collomosse and P. M. Hall. Cubist style rendering from photographs. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):443–453, 2003.

[40] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 207–212, New York, NY, USA, 1984. ACM.

[41] Florin Cutzu, Riad Hammoud, and Alex Leykin. Estimating the photorealism of images: Distinguishing paintings from photographs. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–305. IEEE, 2003.

[42] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, page 736–744, New York, NY, USA, 2002. Association for Computing Machinery.

[43] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, 2001.

[44] Felix & Paul Studios. Wild: The experience, 2015. https://www.felixandpaul.com/?projects/wild.

[45] James A Ferwerda. Three varieties of realism in computer graphics. In *Human vision and electronic imaging viii*, volume 5007, pages 290–297. SPIE, 2003.

[46] Matthew Flagg, Atsushi Nakazawa, Qiushuang Zhang, Sing Bing Kang, Young Kee Ryu, Irfan Essa, and James M. Rehg. Human video textures. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 199–206, New York, NY, USA, 2009. ACM.

[47] Lester Randolph Ford Jr. and Delbert Ray Fulkerson. *Flows in networks*. RAND Corporation, Santa Monica, CA, August 1962. Report number R-375-PR.

[48] Elodie Fourquet. Composition in perspectives. In *Proceedings of the Fourth Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 9–16, 2008.

[49] Ohad Fried, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. Perspective-aware manipulation of portrait photos. *ACM Transactions on Graphics (TOG)*, 35(4):1–10, 2016.

[50] Ohad Fried, Ayush Tewari, Michael Zollhöfer, Adam Finkelstein, Eli Shechtman, Dan B Goldman, Kyle Genova, Zeyu Jin, Christian Theobalt, and Maneesh Agrawala. Text-based editing of talking-head video. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[51] Raghudeep Gadde, Qianli Feng, and Aleix M Martinez. Detail me more: Improving gan's photo-realism of complex scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13950–13959, 2021.

[52] Songwei Ge, Taesung Park, Jun-Yan Zhu, and Jia-Bin Huang. Expressive text-to-image generation with rich text, 2023.

[53] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, oct 2020.

[54] Google ATAP. Google spotlight stores, 2019. https://atap.google.com/spotlight-stories/.

[55] Steve Grogorick, Georgia Albuquerque, and Marcus A. Magnor. Comparing unobtrusive gaze guiding stimuli in head-mounted displays. In *2018 IEEE International Conference on Image Processing, ICIP 2018, Athens, Greece, October 7-10, 2018*, pages 2805–2809, Athens, Greece, 2018. IEEE.

[56] Steve Grogorick, Michael Stengel, Elmar Eisemann, and Marcus Magnor. Subtle gaze guidance for immersive environments. In *Proceedings of the ACM Symposium on Applied Perception*, SAP '17, pages 4:1–4:7, New York, NY, USA, 2017. ACM.

[57] Jan Gugenheimer, Dennis Wolf, Gabriel Haas, Sebastian Krebs, and Enrico Rukzio. A demonstration of swivrchair: A motorized swivel chair to nudge users' orientation for 360 degree storytelling in virtual reality. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, pages 281–284, New York, NY, USA, 2016. ACM.

[58] Hajime Hata, Hideki Koike, and Yoichi Sato. Visual guidance with unnoticed blur effect. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, pages 28–35, New York, NY, USA, 2016. ACM.

[59] Mingming He, Jing Liao, Pedro V. Sander, and Hugues Hoppe. Gigapixel panorama video loops. *ACM Trans. Graph.*, 37(1):3:1–3:15, November 2017.

[60] Zhenliang He, Wangmeng Zuo, Meina Kan, Shiguang Shan, and Xilin Chen. Attgan: Facial attribute editing by only changing what you want. *IEEE transactions on image processing*, 28(11):5464–5478, 2019.

[61] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control, 2022.

[62] Darragh Higgins, Donal Egan, Rebecca Fribourg, Benjamin Cowan, and Rachel McDonnell. Ascending from the valley: Can state-of-the-art photorealism avoid the uncanny? In *ACM symposium on applied perception 2021*, pages 1–5, 2021.

[63] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[64] hreniuca. Bride and groom embracing in paris. https://stock.adobe.com/images/bride-and-groom-embracing-in-paris/81288729.

[65] Shi-Min Hu, Fang-Lue Zhang, Miao Wang, Ralph R Martin, and Jue Wang. Patchnet: A patch-based image representation for interactive library-driven image editing. *ACM Transactions on Graphics (TOG)*, 32(6):1–12, 2013.

[66] Corneliu Ilisescu, Halil Aytac Kanaci, Matteo Romagnoli, Neill D. F. Campbell, and Gabriel J. Brostow. Responsive action-based video synthesis. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 6569–6580, New York, NY, USA, 2017. ACM.

[67] Ingusk. Young girl standing by the tian tan buddha, big buddha in hong kong. https://stock.adobe.com/images/young-girl-standing-by-the-tian-tan-buddha-big-buddha-in-hong-kong/189942971.

[68] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[69] Varun Jampani, Huiwen Chang, Kyle Sargent, Abhishek Kar, Richard Tucker, Michael Krainin, Dominik Kaeser, William T. Freeman, David Salesin, Brian Curless, and Ce Liu. Slide: Single image 3d photography with soft layering and depth-aware inpainting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12518–12527, October 2021.

[70] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[71] Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. Cliplets: Juxtaposing still and dynamic imagery. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 251–260, New York, NY, USA, 2012. ACM.

[72] Hyunyoung Jung, Eunhyeok Park, and Sungjoo Yoo. Fine-grained semantics-aware representation enhancement for self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12642–12652, October 2021.

[73] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, 1990.

[74] Hyeongwoo Kim, Pablo Garrido, Ayush Tewari, Weipeng Xu, Justus Thies, Matthias Niessner, Patrick Pérez, Christian Richardt, Michael Zollhöfer, and Christian Theobalt. Deep video portraits. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[75] Jan Koenderink, Andrea Doorn, Robert Pepperell, and Baingio Pinna. On right and wrong drawings. *Art & Perception*, 4:1–38, 02 2016.

[76] Johannes Kopf, Kevin Matzen, Suhib Alsisan, Ocean Quigley, Francis Ge, Yangming Chong, Josh Patterson, Jan-Michael Frahm, Shu Wu, Matthew Yu, Peizhao Zhang, Zijian He, Peter Vajda, Ayush Saraf, and Michael Cohen. One shot 3d photography. 39(4), 2020.

[77] Jed Lengyel. The convergence of graphics and vision. *Computer*, 31(7):46–53, 1998.

[78] Philippe Levieux, James Tompkin, and Jan Kautz. Interactive viewpoint video textures. In *Proceedings of the 9th European Conference on Visual Media Production*, CVMP '12, pages 11–17, New York, NY, USA, 2012. ACM.

[79] Shuai Li, Jiaying Shi, Wenfeng Song, Aimin Hao, and Hong Qin. Hierarchical object relationship constrained monocular depth estimation. *Pattern Recognition*, 120:108116, 2021.

[80] Jing Liao, Mark Finch, and Hugues Hoppe. Fast computation of seamless video loops. *ACM Trans. Graph.*, 34(6):197:1–197:10, October 2015.

[81] Zicheng Liao, Neel Joshi, and Hugues Hoppe. Automated video looping with progressive dynamism. *ACM Trans. Graph.*, 32(4):77:1–77:10, July 2013.

[82] Yen-Chen Lin, Yung-Ju Chang, Hou-Ning Hu, Hsien-Tzu Cheng, Chi-Wen Huang, and Min Sun. Tell me where to look: Investigating ways for assisting focus in 360° video. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2535–2545, New York, NY, USA, 2017. ACM.

[83] Yung-Ta Lin, Yi-Chi Liao, Shan-Yuan Teng, Yi-Ju Chung, Liwei Chan, and Bing-Yu Chen. Outside-in: Visualizing out-of-sight regions-of-interest in a 360° video using spatial picture-in-picture previews. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 255–265, New York, NY, USA, 2017. ACM.

[84] Andrew Lippman. Movie-maps: An application of the optical videodisc to computer graphics. *SIGGRAPH Comput. Graph.*, 14(3):32–42, July 1980.

[85] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proc. ECCV*, 2018.

[86] Hongyu Liu, Ziyu Wan, Wei Huang, Yibing Song, Xintong Han, and Jing Liao. Pd-gan: Probabilistic diverse gan for image inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9371–9381, June 2021.

[87] Sean J. Liu, Maneesh Agrawala, Stephen DiVerdi, and Aaron Hertzmann. View-dependent video textures for 360° video. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19. ACM, 2019.

[88] Sean J. Liu, Maneesh Agrawala, Stephen DiVerdi, and Aaron Hertzmann. ZoomShop: Depth-Aware Editing of Photographic Composition. *Computer Graphics Forum*, 2022.

[89] CH Lo and Alan Chalmers. Stereo vision for computer graphics: the effect that stereo vision has on human judgments of visual realism. In *Proceedings of the 19th spring conference on Computer graphics*, pages 109–117, 2003.

[90] Helwig Löffelmann and Eduard Gröller. Ray tracing with extended cameras. *The Journal of Visualization and Computer Animation*, 7(4):211–227, 1996.

[91] Xin Ma, Xiaoqiang Zhou, Huaibo Huang, Zhenhua Chai, Xiaolin Wei, and Ran He. Free-form image inpainting via contrastive attention network. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 9242–9249. IEEE, 2021.

[92] Will Marlow. View from the top of the notre dame in paris, July 2012. https://www.flickr.com/photos/williammarlow/7643827866. Original photo licensed under CC BY-NC-SA 2.0.

[93] Brian Matiash. Add impact to your photos with free transform in adobe photoshop, 2021. https://petapixel.com/2021/04/12/add-impact-to-your-photos-with-free-transform-in-adobe-photoshop/.

[94] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[95] S. Mahdi H. Miangoleh, Sebastian Dille, Long Mai, Sylvain Paris, and Yagiz Aksoy. Boosting monocular depth estimation models to high-resolution via content-adaptive multi-resolution merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9685–9694, June 2021.

[96] Mistervlad. Apollo fountain in versailles gardens, paris, france. https://stock.adobe.com/images/apollo-fountain-in-versailles-gardens-paris-france/214090987.

[97] Vaidotas Mišeikis. Faro de formentor, July 2011. https://www.flickr.com/photos/v4idas/6385364319. Original photo licensed under CC BY-NC-ND 2.0.

[98] Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. Dreamix: Video diffusion models are general video editors. *arXiv preprint arXiv:2302.01329*, 2023.

[99] National Geographic. Lions 360°, 2017. https://www.youtube.com/watch?v=sPyAQQklc1s.

[100] Netflix. Stranger things: Virtual reality / 360 experience, 2016. https://www.youtube.com/watch?v=yg29RvYNSDQ.

[101] Duc Quang Nguyen, Ronald Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 721–728, 2002.

[102] Lasse T. Nielsen, Matias B. Møller, Sune D. Hartmeyer, Troels C. M. Ljung, Niels C. Nilsson, Rolf Nordahl, and Stefania Serafin. Missing the point: An exploration of how to guide users' attention during cinematic virtual reality. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, VRST '16, pages 229–232, New York, NY, USA, 2016. ACM.

[103] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3d ken burns effect from a single image. *ACM Transactions on Graphics (TOG)*, 38(6):1–15, 2019.

[104] Xingang Pan, Ayush Tewari, Thomas Leimkühler, Lingjie Liu, Abhimitra Meka, and Christian Theobalt. Drag your gan: Interactive point-based manipulation on the generative image manifold, 2023.

[105] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. *Advances in Neural Information Processing Systems*, 33:7198–7211, 2020.

[106] Gaurav Parmar, Krishna Kumar Singh, Richard Zhang, Yijun Li, Jingwan Lu, and Jun-Yan Zhu. Zero-shot image-to-image translation, 2023.

[107] Peter J. Passmore, Maxine Glancy, Adam Philpot, Amelia Roscoe, Andrew Wood, and Bob Fields. Effects of viewing condition on user experience of panoramic video. In *Proceedings of the 26th International Conference on Artificial Reality and Telexistence and the 21st Eurographics Symposium on Virtual Environments*, ICAT-EGVE '16, pages 9–16, Goslar Germany, Germany, 2016. Eurographics Association.

[108] Amy Pavel, Björn Hartmann, and Maneesh Agrawala. Shot orientation controls for interactive cinematography with 360 video. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 289–297, New York, NY, USA, 2017. ACM.

[109] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016.

[110] Yael Pritch, Eitam Kav-Venaki, and Shmuel Peleg. Shift-map image editing. In *2009 IEEE 12th international conference on computer vision*, pages 151–158. IEEE, 2009.

[111] Ed Quigley, Yue Yu, Jingwei Huang, Winnie Lin, and Ronald Fedkiw. Real-time interactive tree animation. *IEEE transactions on visualization and computer graphics*, 24(5):1717–1727, 2017.

[112] Paul Rademacher, Jed Lengyel, Edward Cutrell, and Turner Whitted. Measuring the perception of visual realism in images. In *Rendering Techniques 2001: Proceedings of the Eurographics Workshop in London, United Kingdom, June 25–27, 2001 12*, pages 235–247. Springer, 2001.

[113] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[114] Boris V Rauschenbach. Perceptual perspective and cezanne's landscapes. *Leonardo*, 15(1):28–33, 1982.

[115] Erik Reinhard, Alexei A Efros, Jan Kautz, and Hans-Peter Seidel. On visual realism of synthesized imagery. *Proceedings of the IEEE*, 101(9):1998–2007, 2013.

[116] Stephan R Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1700–1715, 2022.

[117] Rocksteady Studios. Batman: Arkham vr, 2016. http://rocksteadyltd.com/#arkham-vr.

[118] Christoph Alexander Rosenberg. Over there! visual guidance in 360-degree videos and other virtual environments. Master's thesis, Universität des Saarlandes, September 2017.

[119] Steve Rubin and Maneesh Agrawala. Generating emotionally relevant musical scores for audio stories. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 439–448, New York, NY, USA, 2014. ACM.

[120] 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*, SIGGRAPH '00, pages 489–498, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[121] Robert Sedgewick. *Algorithms in C++ Part 5: Graph Algorithms (3rd Edition)*. Addison-Wesley Professional, Reading, Massachusetts, August 2001.

[122] Vidya Setlur, Saeko Takagi, Ramesh Raskar, Michael Gleicher, and Bruce Gooch. Automatic image retargeting. In *Proceedings of the 4th international conference on Mobile and ubiquitous multimedia*, pages 59–68, 2005.

[123] Jun'ichiro Seyama and Ruth S Nagayama. The uncanny valley: Effect of realism on the impression of artificial human faces. *Presence*, 16(4):337–351, 2007.

[124] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image, 2019.

[125] Thomas K Sharpless, Bruno Postle, and Daniel M German. Pannini: A new projection for renderingwide angle perspective images. In *Computational Aesthetics*, pages 9–16, 2010.

[126] Alia Sheikh, Andy Brown, Zillah Watson, and Michael Evans. Directing attention in 360-degree video. In *IBC 2016 Conference*, pages 29–37, Amsterdam, Netherlands, September 2016. IBC.

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

[128] YiChang Shih, Wei-Sheng Lai, and Chia-Kai Liang. Distortion-free wide-angle portraits on camera phones. *ACM Trans. Graph.*, 38(4), July 2019.

[129] Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. Summarizing visual data using bidirectional similarity. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[130] Karan Singh. A fresh perspective. In *Graphics interface*, volume 2002, pages 17–24. Citeseer, 2002.

[131] Travis Stebbins and Eric D. Ragan. Redirecting view rotation in immersive movies with washout filters. In *Proceedings of the IEEE Virtual Reality*, Osaka, Japan, 2019. IEEE.

[132] Sara L Su, Frédo Durand, and Maneesh Agrawala. De-emphasis of distracting image regions using texture power maps. 2005.

[133] Zhaolin Su and Shigeo Takahashi. Real-time enhancement of image and video saliency using semantic depth of field. In *International Conference on Computer Vision Theory and Applications (VISAPP) (2)*, pages 370–375, Angers, France, 2010. INSTICC.

[134] Bongwon Suh, Haibin Ling, Benjamin B Bederson, and David W Jacobs. Automatic thumbnail cropping and its effectiveness. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 95–104, 2003.

[135] Maitreya Suin, Kuldeep Purohit, and A. N. Rajagopalan. Distillation-guided image inpainting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2481–2490, October 2021.

[136] Pedro Szekely. Monument valley, August 2014. https://www.flickr.com/photos/pedrosz/35250460535. Modified from original and used under CC BY-SA 2.0. Derivatives are licensed under CC BY-SA 4.0.

[137] Pedro Szekely. Venice, italy, October 2014. https://www.flickr.com/photos/pedrosz/38269434695. Modified from original and used under CC BY-SA 2.0. Derivatives are licensed under CC BY-SA 4.0.

[138] Christopher C Tanner, Christopher J Migdal, and Michael T Jones. The clipmap: a virtual mipmap. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 151–158, 1998.

[139] Bernd Thaller. Lake bohinj, August 2016. https://www.flickr.com/photos/bernd_thaller/30816367150. Modified from original and used under CC BY 2.0.

[140] Omer Tov, Yuval Alaluf, Yotam Nitzan, Or Patashnik, and Daniel Cohen-Or. Designing an encoder for stylegan image manipulation. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021.

[141] Wade Tregaskis. Yosemite valley, February 2020. https://www.flickr.com/photos/wadetregaskis/50156486633. Original photo licensed under CC BY-NC 2.0.

[142] Marc Van den Broeck, Fahim Kawsar, and Johannes Schöning. It's all around you: Exploring 360° video viewing experiences on mobile devices. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, pages 762–768, New York, NY, USA, 2017. ACM.

[143] Eduardo E. Veas, Erick Mendez, Steven K. Feiner, and Dieter Schmalstieg. Directing attention and influencing memory with visual saliency modulation. In *Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems, CHI '11, pages 1471–1480, New York, NY, USA, 2011. ACM.

[144] Walt Disney Animation Studios. Cycles. SIGGRAPH Immersive Pavilion, 2018. Dir. Jeff Gipson.

[145] 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.

[146] Yiran Wang, Xingyi Li, Min Shi, Ke Xian, and Zhiguo Cao. Knowledge distillation for fast and accurate monocular depth estimation on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2457–2465, June 2021.

[147] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[148] Huikai Wu, Shuai Zheng, Junge Zhang, and Kaiqi Huang. Gp-gan: Towards realistic high-resolution image blending. In *Proceedings of the 27th ACM international conference on multimedia*, pages 2487–2495, 2019.

[149] Mai Xu, Chen Li, Shanyi Zhang, and Patrick Le Callet. State-of-the-art in 360 video/image processing: Perception, assessment and compression. *IEEE Journal of Selected Topics in Signal Processing*, 14(1):5–26, 2020.

[150] Su Xue, Aseem Agarwala, Julie Dorsey, and Holly Rushmeier. Understanding and improving the realism of image composites. *ACM Transactions on graphics (TOG)*, 31(4):1–10, 2012.

[151] Binxin Yang, Shuyang Gu, Bo Zhang, Ting Zhang, Xuejin Chen, Xiaoyan Sun, Dong Chen, and Fang Wen. Paint by example: Exemplar-based image editing with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18381–18391, 2023.

[152] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Proc. ICCV*, 2019.

[153] Jingyi Yu and Leonard McMillan. A framework for multiperspective rendering. *Rendering Techniques*, 4:61–68, 2004.

[154] Jingyi Yu and Leonard McMillan. General linear cameras. In *European Conference on Computer Vision*, pages 14–27. Springer, 2004.

[155] Lihi Zelnik-Manor and Pietro Perona. Automating joiners. In *Proceedings of the 5th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '07, page 121–131, New York, NY, USA, 2007. Association for Computing Machinery.

[156] Lihi Zelnik-Manor, Gabriele Peters, and Pietro Perona. Squaring the circle in panoramas. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1292–1299. IEEE, 2005.

[157] Yu Zeng, Zhe Lin, Huchuan Lu, and Vishal M. Patel. Cr-fill: Generative image inpainting with auxiliary contextual reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14164–14173, October 2021.

[158] Jingxin Zhang, Eike Langbehn, Dennis Krupke, Nicholas Katzakis, and Frank Steinicke. Detection thresholds for rotation and translation gains in 360° video-based telepresence systems. *IEEE Transactions on Visualization and Computer Graphics*, 24(4):1671–1680, April 2018.

[159] Jinsong Zhang, Kun Li, Yu-Kun Lai, and Jingyu Yang. Pise: Person image synthesis and editing with decoupled gan, 2021.

[160] Yuqian Zhou, Connelly Barnes, Eli Shechtman, and Sohrab Amirghodsi. Transfill: Reference-guided image inpainting by merging multiple color and spatial transformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2266–2276, June 2021.

[161] Jun-Yan Zhu, Philipp Krahenbuhl, Eli Shechtman, and Alexei A Efros. Learning a discriminative model for the perception of realism in composite images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3943–3951, 2015.

[162] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part V 14*, pages 597–613. Springer, 2016.

[163] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[164] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. *Advances in neural information processing systems*, 30, 2017.

[165] Yiming Zhu, Hongyu Liu, Yibing Song, Ziyang Yuan, Xintong Han, Chun Yuan, Qifeng Chen, and Jue Wang. One model to edit them all: Free-form text-driven image manipulation with

semantic modulations. *Advances in Neural Information Processing Systems*, 35:25146–25159, 2022.

[166] Denis Zorin and Alan H Barr. Correction of geometric perceptual distortions in pictures. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 257–264, 1995.