

# ERROR-CONCEALED IMAGE-BASED RENDERING

Der Carl-Friedrich-Gauß Fakultät  
Technische Universität Carola-Wilhelmina zu Braunschweig

zur Erlangung des Grades

Doktor Ingenieur (Dr.-Ing.)

vorgelegte

Dissertation

von **Martin Eisemann**

geboren in **Köln**

am **14. März 1980**

Eingereicht am: 25.03.2011

Disputation am: 06.07.2011

Referent: Prof. Dr.-Ing. Marcus Magnor

Koreferent: Prof. Dr.-Ing. Jan Kautz

(2011)



## ABSTRACT

---

Creating photo-realistic images has been one of the major goals in computer graphics since its early days. Instead of modeling the complexity of nature with standard modeling tools, image-based approaches aim at exploiting real-world footage directly, as they are photo-realistic by definition. A drawback of these approaches has always been that the composition or combination of different sources is a non-trivial task, often resulting in annoying visible artifacts. In this thesis we focus on different techniques to diminish visible artifacts when combining multiple images in a common image domain. The results are either novel images, when dealing with the composition task of multiple images, or novel video sequences rendered in real-time, when dealing with video footage from multiple cameras.

## KURZFASSUNG

---

Fotorealismus ist seit jeher eines der großen Ziele in der Computergrafik. Anstatt die Komplexität der Natur mit standardisierten Modellierungswerkzeugen nachzubauen, gehen bildbasierte Ansätze den umgekehrten Weg und verwenden reale Bildaufnahmen zur Modellierung, da diese bereits per Definition fotorealistisch sind. Ein Nachteil dieser Variante ist jedoch, dass die Komposition oder Kombination mehrerer Quellbilder eine nicht-triviale Aufgabe darstellt und häufig unangenehm auffallende Artefakte im erzeugten Bild nach sich zieht. In dieser Dissertation werden verschiedene Ansätze verfolgt, um Artefakte zu verhindern oder abzuschwächen, welche durch die Komposition oder Kombination mehrerer Bilder in einer gemeinsamen Bild-domäne entstehen. Im Ergebnis liefern die vorgestellten Verfahren neue Bilder oder neue Ansichten einer Bildsammlung oder Videosequenz, je nachdem, ob die jeweilige Aufgabe die Komposition mehrerer Bilder ist oder die Kombination mehrerer Videos verschiedener Kameras darstellt.



## SUMMARY

---

Computer graphics is a large field of computer science that has received a lot of attention during the last decades due to its success in the movie, games and entertainment industry. The ever-rising demands for realism in these application fields resulted in a huge leap in complexity of models and scene representation. This, in turn, leads to a variety of new challenges to overcome, be it in acquisition, modeling, post production or rendering. The direction taken by industry is currently to invest enough money, time and manual labor in order to achieve the desired results. Twentieth Century Fox invested 237,000,000 U.S. Dollars in James Cameron's *Avatar* in 2009 [38], i.e., more than 24,000\$ per second, obviously only few companies can afford to follow this trend.

Image-based rendering techniques promise to be a cost-effective alternative by exploiting photo and video footage directly. Since these are photo-realistic by definition, photo-realism is no direct concern. But it turns out that high-quality image-based rendering results require also a lot of hardware and precise setups. For convincing results, hundreds of perfectly calibrated input cameras may be needed even for small objects or relatively simple scenes [146]. Therefore, the main problem of monetary costs, necessary time and amount of manual labor, is only shifted towards the costs of material expenses and time needed for the camera and scene setup.

To reduce the number of cameras needed, a step from pure image-based rendering to geometry guided image-based rendering can be taken. 3D scene reconstruction algorithms can provide approximate representations of the original scene geometry to facilitate rendering from fewer cameras. But as image-based rendering has also become popular in the field of sports events, additional constraints like real-time performance might have to be taken into account. To handle this requirement, better and faster hardware is needed, which brings one back to the monetary problem. An alternative are faster reconstruction algorithms, which, however, go hand in hand with lower rendering quality. Visible errors emerge as the image reconstruction problem becomes more difficult.

The same problem of visible artifacts does not only occur in movie productions or broadcasting. In fact, any image-based rendering technique irrevocably suffers from incomplete or low quality input data, whether it is free-viewpoint video [40], panorama imaging [236] or even texture synthesis [261]. Manual

reworks become necessary again, requiring skilled, and well-paid artists. The main challenge is to find new, efficient ways to achieve high-quality renderings requiring fewer hardware, less manual labor and sometimes even additional constraints, like real-time performance.

The work presented in this thesis addresses these problems and limitations in several fields of image-based rendering. In the beginning we present a new technique for image upsampling and multiscale panoramas from insufficient input images. Additional input images providing higher detail for certain regions, but taken with different cameras, different white balancing or color aberrations, as well as potential structural mismatches, are seamlessly blended with the low resolution panorama image. A detail transfer and enhancement mechanism is provided for regions where no specific details are otherwise available. Further, an easy and flexible rendering scheme for even larger zoom factors and real-time applications is introduced.

In the next part we present an easy-to-use video matting approach that allows even inexperienced users to create high-quality mattes. For certain scenes our video matting system is even able to create foreground mattes for videos without knowledge about the fore- and background and without any user interaction at all.

In the last part of the thesis we deal with known deficiencies in free-viewpoint video. If too few input cameras are provided or the scene reconstruction is imprecise, visible artifacts seem to be inevitable. We investigate the source of these errors in detail and derive two different approaches to diminish the artifacts and create higher quality renderings with fewer cameras, small camera calibration errors and imprecise 3D reconstruction. Both are real-time capable and are therefore applicable to any image-based rendering technique based on multiview projective texture mapping.

## ZUSAMMENFASSUNG

---

Die *Computer Graphik* ist ein weites Feld in der Informatik, welches insbesondere durch seine Anwendung in der Film-, Unterhaltungs- und Spieleindustrie große Aufmerksamkeit erlangt hat. Die steigenden Ansprüche an den Realismus sorgten für einen immensen Komplexitätszuwachs der Modelle und Szenenrepräsentationen. Dies wiederum bedingt verschiedenste neue Probleme, die es zu lösen gilt, sei es in der Akquisition, Modellierung, Nachbearbeitung oder der Darstellung. Die Industrie verfolgt dabei momentan noch den Ansatz nur genügend Geld, Zeit und Arbeitskraft zu investieren, um das gewünschte Ergebnis zu erzielen. Twentieth Century Fox investierte 237,000,000 U.S. Dollars in James Camerons Avatar in 2009 [38], d.h. mehr als 24,000\$ pro Sekunde. Offensichtlich können sich solch einen Aufwand nur wenige Firmen leisten.

Bildbasierte Darstellungsverfahren können eine kostengünstige Alternative anbieten, indem sie aufgenommenes Foto- und Videomaterial direkt zur Darstellung verwenden können. Da diese bereits von der Definition her fotorealistisch sind, ist Realismus kein direktes Problem mehr. Leider hat es sich gezeigt, dass qualitativ hochwertige, bildbasierte Darstellungsverfahren auch eine Menge an Hardware und präzise durchgeführte Einstellungen benötigen. Für überzeugende Ergebnisse sind oft hunderte, perfekt kalibrierte Kameras notwendig, selbst für schmale Objekte oder relativ simple Szenen [146]. Das Hauptproblem der Kosten, Zeit und Arbeitskraft wird dadurch zumeist lediglich umgelegt auf die Materialkosten und Zeit, welche benötigt werden für Kameras und die Szeneneinstellungen.

Durch Verwendung von geometrieunterstützten bildbasierten Darstellungsverfahren kann die Anzahl benötigter Kameras verringert werden. 3D Rekonstruktionsalgorithmen liefern eine approximierete Repräsentation der original Szenengeometrie um die Darstellung auch mit weniger Kameras zu ermöglichen. Da aber bildbasierte Verfahren auch gerade im Sportbereich immer mehr Anklang finden, kommen zusätzliche Anforderungen, wie etwa Echtzeitfähigkeit, hinzu. Um diesen Anforderungen nachzukommen, wird bessere und schneller Hardware benötigt, was uns wieder zum ursprünglichen finanziellen Problem zurückführen würde. Eine Alternative würden auch schnellere Rekonstruktionsalgorithmen liefern, welche jedoch Hand in Hand mit geringerer Qualität in der Darstellung gehen. Sichtbare Artefakte treten auf, da die Bildrekonstruktion entsprechend schwerer wird.

Artefakte im Rekonstruktionsergebnis tauchen nicht nur in der Film- und Fernsehproduktion auf. Tatsache ist, dass jedweder bildbasierte Ansatz in seiner Qualität leidet, sobald er es mit zu wenigen Daten oder Daten zu geringer Qualität zu tun hat. Sei es Free-Viewpoint Video [40], Panoramafotografie [236] oder selbst Textursynthese [261]. Kostspielige, manuelle Nachbearbeitung wird somit wieder notwendig.

Die große Herausforderung ist es also neue, effiziente Wege zu finden um qualitativ hochwertige Darstellungen mit weniger Hardware, weniger Handarbeit und manchmal selbst zusätzlichen Herausforderungen, wie Echtzeitdarstellung, zu erzeugen.

Die in dieser Dissertation vorgestellten Arbeiten gehen diese Probleme und Limitierungen in verschiedensten Bereichen der bildbasierten Darstellung an. Zunächst beschäftigen wir uns mit dem Problem des Hochskalierens für digitale Bilder und Panoramaaufnahmen aus unzureichenden Eingabedaten. Zusätzliche Bilder, welche einen höheren Detailgrad aufweisen für bestimmte Bereiche der aufgenommenen Szene, werden nahtlos in das niedriger aufgelöste Panorama eingebunden. Schwierigkeiten entstehen dabei durch unterschiedliche Kameramodelle, Weißabgleich oder Farbabweichungen, sowie struktureller Diskrepanzen. Ein Detailtransfer sorgt zudem für mehr Details in Bildregionen, für die ansonsten keine passenden Eingabebilder gefunden werden konnten. Zudem wird ein flexibles Verfahren vorgestellt für die Echtzeitdarstellung von noch größeren Zooms in Bilder hinein.

Im darauffolgenden Abschnitt wird ein einfach anzuwendendes Matting Verfahren zur Trennung von Vorder- und Hintergrund in Videos vorgestellt, welches es auch unerfahrenen Benutzern erlaubt qualitativ hochwertige Mattes zu erstellen. Für manche Szenen kann der vorgestellte Algorithmus sogar die Mattes für komplette Videos ohne weiteres Zutun oder Wissen über den Hintergrund erstellen.

Im letzten Teil dieser Arbeit wird auf bekannte Schwierigkeiten und Unzulänglichkeiten in Free-Viewpoint Video Applikationen eingegangen. Sind zu wenige Eingabebilder für eine präzise Szenenrekonstruktion gegeben, sind sichtbare Artefakte unvermeidbar. Wir untersuchen detailliert die Ursprünge dieser Artefakte und leiten daraus zwei unterschiedliche Ansätze zu ihrer Vermeidung ab, um somit qualitativ verbesserte Bilddarstellungen zu erzeugen, trotz weniger Kameras, kleinerer Kalibrierungsfehler und ungenau rekonstruierter 3D Geometrie. Beide Ansätze sind Echtzeitfähig und für alle bildbasierten Ansätze einsetzbar, welche auf projektiver Texturierung mit mehreren Kameras basieren.



## ACKNOWLEDGMENTS

---

There are so many people I would like to thank and express my gratitude to in these acknowledgements, who supported me and contributed to this dissertations in more ways than they may imagine. First of all, I would like to mention my parents Freya and Hans, my siblings Almuth and Elmar, for their support and love, also my grandma Hadumuth (yes, this name really exists), my nephew Finn and of course Myriam. She always supported me, encouraged me at all times and even endured my sometimes grouchy mood during the time of writing this thesis.

I would like to thank my supervisor Marcus Magnor, who not only supported my research but before anything else gave me the opportunity to work here at the Computer Graphics Lab of the Technische Universität Braunschweig and therefore made me meet so many wonderful people throughout the years. I have never worked in a more friendly, collaborative and familial environment and will never forget all those funny happenings here.

I would like to thank Timo Stich for always showing me the joy of science and for extending me such a warm welcome in Braunschweig. And I want to thank Christian Linz and Georgia Albuquerque who shared the office with me and never complained when I disrupted their thoughts. For this I should also express a lot of my gratitude to Anita Sellent, our Ms. Mathematician. But they were not the only ones, thank you Stephan Wenger, Christian Lipski, Felix Klose, Lorenz Rogge, Kai Berger, Thomas Neumann, Lea Lindemann, Kai Ruhl and Benjamin Meyer for the helpful scientific discussions. Thank you Anja Franzmeier for keeping all of the administrative part as simple as possible for me and all of us. Therefore I should also mention Christin Wähner, Markus Galda, Yasemin Yueksel-Glogowski, Benjamin Flecken, Kristina Branz, Florian Barucha, Brian Schimmel, Arthur Martens and Patrick McLaren. Thank you Carsten for keeping our computers running and your trenchant sense of humor hitting the nail on the head in so many situations. And thank you all for the fun we had and hopefully will have in the future.



# CONTENTS

---

<b>I</b>	<b>Introduction</b>	<b>1</b>
1	INTRODUCTION	3
2	PREREQUISITES	7
2.1	A Generic Image-based Rendering Pipeline . . . . .	7
2.2	The Plenoptic Function . . . . .	8
2.3	Image Formation . . . . .	9
2.4	Spatial Transformations . . . . .	9
2.5	The Camera Model . . . . .	13
2.6	Image Blending . . . . .	15
2.7	Image morphing . . . . .	15
2.8	3D Reconstruction . . . . .	16
2.9	Free Viewpoint Video . . . . .	20
2.10	Optical Flow . . . . .	22
2.11	Matting . . . . .	23
2.12	Gradient Domain Compositing . . . . .	24
2.13	Exemplar-based Texture Synthesis . . . . .	26
<b>II</b>	<b>Error Concealment in Seamless Image Synthesis</b>	<b>29</b>
3	INTRODUCTION	31
3.1	Background . . . . .	31
3.2	Related Work . . . . .	32
4	PHOTO ZOOM	39
4.1	Introduction . . . . .	39
4.2	Dependency Graph Construction . . . . .	41
4.3	Detail Transfer . . . . .	43
4.4	Constrained Multiscale Detail Synthesis . . . . .	51
4.5	Results . . . . .	56
4.6	Discussion . . . . .	63
5	ZIPMAPS: ZOOM-INTO-PARTS TEXTURE MAPS	67
5.1	Introduction . . . . .	67
5.2	Zipmaps . . . . .	68
5.3	Results . . . . .	72
5.4	Discussion . . . . .	73

<b>III</b>	<b>Error Concealment in Video Matting</b>	<b>77</b>
6	INTRODUCTION	79
6.1	Background . . . . .	79
6.2	Related Work . . . . .	80
7	SPECTRAL VIDEO MATTING	83
7.1	Introduction . . . . .	83
7.2	Spectral Matting . . . . .	84
7.3	Spectral Video Matting . . . . .	86
7.4	Results . . . . .	88
7.5	Discussion . . . . .	89
<b>IV</b>	<b>Error Concealment in Image-based Rendering</b>	<b>91</b>
8	INTRODUCTION	93
8.1	Background . . . . .	93
8.2	Related Work . . . . .	96
9	ERROR ANALYSIS	103
9.1	Introduction . . . . .	103
9.2	Problem Description . . . . .	104
9.3	A Geometric Analysis of Ghosting Artifacts . . . . .	107
10	FILTERED BLENDING FOR MULTIVIEW PROJECTIVE TEXTURING	115
10.1	Introduction . . . . .	115
10.2	View-dependent Ghosting Artifact Analysis . . . . .	116
10.3	View-dependent Filtering . . . . .	117
10.4	GPU Implementation . . . . .	118
10.5	Results . . . . .	121
10.6	Discussion . . . . .	122
11	FLOATING TEXTURES	127
11.1	Introduction . . . . .	127
11.2	Floating Textures . . . . .	129
11.3	Soft Visibility . . . . .	132
11.4	GPU Implementation . . . . .	134
11.5	Results . . . . .	136
11.6	Discussion . . . . .	139
<b>V</b>	<b>Conclusion</b>	<b>141</b>
12	SUMMARY	143
12.1	Future Work . . . . .	144

<b>VI Appendix</b>	<b>147</b>
A NOTATION	149
B PHOTO CREDITS	151
Bibliography	153



Part I.  
Introduction





INTRODUCTION

---

*I was afraid they would give me a math test to get in.  
I was even more afraid they would give me a math test to get out!*

— Don Marinelli

Images represent the fundamental basis of any visual research. Computer vision focuses on images as input data with the aim to transform the contained information into a new representation useful for tasks such as motion tracking, object recognition or scene reconstruction. On the other hand, computer graphics traditionally generates images as the output of its processing pipeline, e.g. in data visualization, computer animation or simply to synthesize new views for a geometric scene description, used in computer games or virtual environments. However, in the last two decades computer graphics evolved into a new direction by making use of images also as input data to its algorithms. This provided new, exciting ways to create (photo-realistic) renderings. Examples are image-based rendering techniques [59, 146, 153], or the classic discipline of image compositing [34, 189].

Compositing can be summarized as combining two or more images into a single output image, similar to collages. Almost any high-quality movie production that incorporates computer generated content nowadays relies on this concept for a more efficient workflow [34]. The most simple, though most heavily used variant in movie production is the composition of different images without changing their respective content. This can be seen as a simple layering concept. Different images or image patches are drawn on top of each other by simply painting over the underlying content or by blending using an alpha mask to describe the opacity of each layer. While the composition itself is a rather simple task once the alpha mask has been created, the preceding steps require more attention, especially the object extraction, sometimes also referred to as matting or roto-scoping [253]. Extracting an object pixel-wise by hand can be tedious enough for a single image, but for longer videos it would become overwhelming. For more complex or semi-transparent objects, like hair, it would be even impossible as pixel-wise copying could never extract a realistic matte. As transparent objects are always a combination of the foreground and background

color, the matting problem becomes one of reconstructing the respective colors as well as the transparency of the object. While the task is manageable for simple backgrounds, such as a blue screen [218], it evolves to a very complex task for natural backgrounds [144, 145] and an even more complex task for videos [17, 46]. Most research in this field of compositing therefore aims at simplifying or reducing the workload of the artist to accomplish his or her object extraction task. Unfortunately, most of the algorithms either rely only on color statistics [203], which require controlled environments for good results, or they lack the necessary robustness resulting in the necessity for a lot of user interaction [46].

Another variant of the previously mentioned compositing is the seamless integration of image patches (source) into another image (target) [184]. In this variant, the content of the source is adjusted in a way to preserve its overall structure and to seamlessly merge with the underlying content of the target. The goal is to convince the viewer that he is looking at a single, realistic image, in which he can no longer differentiate between the different sources. In order to create realistic transitions between a source and target image, structural mismatches between both need to be removed. In some cases one even has to deal with frequency mismatches, e.g., if the source or target does not convey as high-frequency information in comparison to the respective counterpart [234]. Again this can create visible seams between the source and target which one has to deal with, either by hiding the seams or, as we do in this thesis, by adding new, compatible high-frequency information to the lower frequency part of the image, i.e., new textural information has to be hallucinated by some plausible means.

Generalizing the concept of image compositing, we can find it in other fields of image-based computer graphics as well, e.g. at the borderline between vision and graphics, namely in multi-view image-based rendering, e.g. [40, 59, 146]. In this field an essential requirement is the realistic reproduction of the input data, with regard to plausibility instead of physical correctness. Application examples are image morphing [23] to interpolate between two views, or free-viewpoint video [40], where new images are created on the basis of a freely movable virtual camera. The classic approach is to transfer the input images into the output image domain and combine/composite them in a meaningful and plausible manner. One application example, which is already used by the industry, is the analysis of sports events with changing viewpoints [115]. But imprecisions in the scene reconstruction or camera calibration can lead to visually disturbing artifacts.

This dissertation investigates several representative problems of the spectrum of image-based computer graphics in the context of image compositing:

- Seamless image compositing, upsampling and texture hal-lucination dealing with several artifact-revealing aspects, including color, content mismatch and frequency differences;
- Video matting for complex objects;
- Error concealment in image-based rendering techniques which are based on projective texture mapping.

#### THESIS STRUCTURE AND CONTRIBUTION

Parts of this dissertation have already been presented at various conferences including the Eurographics conference, Graphics Interface and the Vision, Modeling and Visualization workshop and have been published in the according conference proceedings [69, 71, 73, 74, 77], journals [72], books [76] and different technical reports [67, 68, 70, 75].

The basis of this dissertation is founded on these publications, but combines them under the unifying concept of error-concealed rendering. After a short introduction and an overview of the necessary background in the first part of this thesis, we examine the problems occurring in seamless image and content synthesis. The main contributions of this second part are listed in the following.

- A system to automatically construct high-resolution images from an unordered set of low resolution photos is presented in Chapter 4. It consists of an automatic pre-processing step to establish correspondences between any number of given photos. The user may then choose one image, and the algorithm automatically creates a higher resolution result, several octaves larger, up to the desired resolution. Detail information is seamlessly added from the other photographs, dealing with structural inconsistencies, color aberrations and frequency mismatches. The applied recursive creation scheme allows to transfer specific details at subpixel positions of the original image.
- In Chapter 5 we present an easy, flexible and hierarchical representation to render detailed texture patches into a classic texture map of limited resolution. Instead of saving a single high-resolution texture map, a single low-resolution texture map is saved, and accompanying high-detail patches are rendered at the interesting positions to provide additional high-resolution content. This gives the opportunity to render different texture patches on top of each other

without any artifacts such as z-fighting, aliasing artifacts, or visible seams between the patches.

In the third part we deal with problems occurring in video matting of complex objects.

- A new, simple-to-use and rapid approach to video matting, the process of pulling a high-quality alpha matte from a video sequence, is presented in Chapter 7. No additional hardware, except for a single camera, is needed, and only very few and intuitive user interactions are required for foreground estimation. For certain scenes the approach is able to estimate the alpha matte for a single video without any user interaction at all.

In the fourth part of the thesis we present new algorithms to deal with errors and artifacts in Free-Viewpoint Video and other image-based rendering techniques.

- An analysis of the causes of artifacts in multiview projective texturing is given in Chapter 9; aliasing as well as global filtering methods are discussed.
- A new graphics-hardware accelerated filtering strategy and a view-dependent definition for ghosting detection to prevent visible artifacts in multiview projective texturing and image-based rendering in real-time is proposed in Chapter 10.
- A new multiview texturing algorithm that warps and blends projected textures at run time to preserve a crisp, detailed texture appearance is presented Chapter 11.
- Both presented methods achieve interactive to real-time frame rates on commodity graphics processing units (GPU). They can be used in combination with many image-based rendering methods or projective texturing applications. Usage of the methods in conjunction with, e.g., visual hull reconstruction [84], light field rendering [146], or free-viewpoint video [40], leads to improved rendering results that are obtained from fewer input images, less accurately calibrated cameras, and coarser 3D geometry proxies.

We conclude in the last part with some thoughts and discussions about the achieved results, draw a conclusion and give an outlook on future work and already published work by others that build on the results of this thesis.

Additionally, to help with the different notations used throughout the thesis we added appendix A on page 149.

## PREREQUISITES

*I have to apologize for the formulae here.  
But these are not mine, so don't blame me.*

— Liang Wang

This thesis touches a variety of different topics in computer graphics. Even though in-depth knowledge for all of these is not necessarily a requirement when reading the thesis, we believe that a brief introduction into the different fields eases understanding.

## 2.1 A GENERIC IMAGE-BASED RENDERING PIPELINE

Figure 1 provides an overview of a generic image-based rendering pipeline. Several images taken from one or multiple cameras serve as input. In the preprocessing step additional information is extracted from the images without altering the images itself, e.g. camera parameters or segmentation masks. The images plus extracted information can then be used to either alter the input images themselves, e.g. for a later composition task, or to reconstruct the underlying 3D geometry of the scene depicted in the images. If all necessary information and images are available, the rendering step combines them in a meaningful way to produce the final output image. Of course, all additional information produced by each of the different steps could be used as input again to the former processing steps.

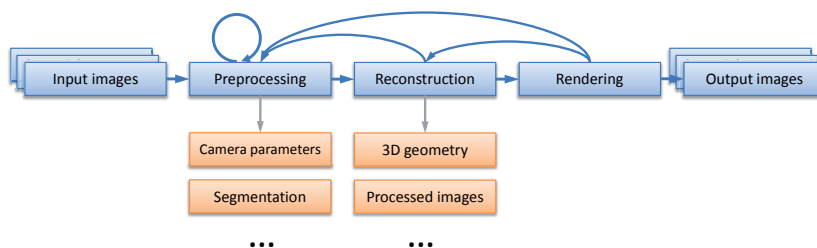


Figure 1.: Generic arrangement of a typical image-based rendering pipeline.

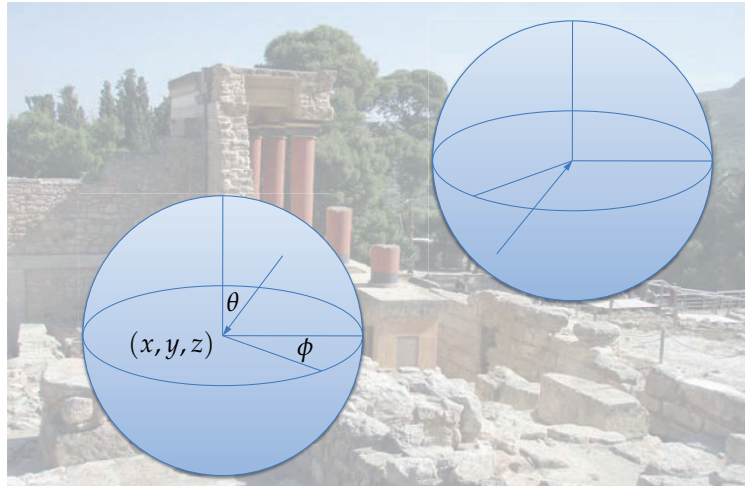


Figure 2.: The plenoptic function describes the angular light distribution for every point in space.

## 2.2 THE PLENOPTIC FUNCTION

Sensing our surrounding world has always been essential to us as humans. Using our sense of sight, hearing, smell, taste and touch we are able to experience our environment and process the incoming information. Not surprisingly, the sense of sight is the most important one for most of us due to our own evolutionary roots. Our eyes serve as sensors capturing the incoming radiance. Classic photo or video cameras are similar sensors used to capture the distribution of light, which can be characterized by the *plenoptic function*:

$$\mathbf{P}(x, y, z, \theta, \phi, t, \lambda) , \quad (2.1)$$

The plenoptic function describes light as a 7D function for every viewpoint  $(x, y, z)$ , viewing direction  $(\theta, \phi)$ , point in time  $t$  and wavelength  $\lambda$  [2]. Most image-based rendering systems deal with a 5D subset of this function, discarding time and wavelengths, Figure 2, and if not stated otherwise we will adopt this simplification throughout this thesis. If the object is assumed to be in a transparent medium, like air, and the viewpoint is placed outside the object's visual hull the plenoptic function can even be reparameterized as a 4D function in ray space [146]. The goal of almost every image-based rendering system is to reconstruct the complete function or parts of it as good as possible, using only the camera calibration data, sometimes a geometry proxy and a set of input images or video. Here, images constitute sparse samples of the plenoptic function.

However, correct estimation of the plenoptic function is not necessarily mandatory in computer graphics. Visual plausibility is usually more important than a physically correct reconstruction. In the different approaches presented in this thesis, we will

not only resample, but change, adjust and hallucinate parts of the plenoptic function.

### 2.3 IMAGE FORMATION

In computer graphics images taken by a digital camera are represented as an array of pixels. Each pixel represents the integral over a small solid angle area of the plenoptic function, described by an  $rgb$  triplet. Therefore, an image can be described as a function  $\mathbf{I} : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}_+^3$ , which assigns to each pixel position  $\mathbf{x} = (x, y) \in \Omega$  a vector  $(r, g, b) \in \mathbb{R}_+^3$ . As the value of the integral saved by a single pixel is assigned to discrete pixel positions in  $\mathbb{N}^2$ , we will assume that color values at any other position  $\notin \mathbb{N}^2$  are determined by bilinear interpolation, i.e. a weighted sum of the four surrounding pixels. We will refer to pixel positions as either  $\mathbf{x}$  or  $(x, y)$ , while the value at a certain pixel is referred to as  $\mathbf{I}(\mathbf{x})$  or  $\mathbf{I}(x, y)$ . If the parameters of the cameras are known, we will sometimes refer to a specific pixel position and its associated value of image  $\mathbf{I}$  as  $\mathbf{I}(x, y, z, \theta, \phi)$  corresponding to the parameters of the plenoptic function  $\mathbf{P}$ . Here  $x, y, z$  are the camera's position in world coordinates. We will also use images as general information buffers to encode, e.g., opacity values or other information. In this case the co-domain of  $\mathbf{I}$  is changed accordingly.

### 2.4 SPATIAL TRANSFORMATIONS

In this section we will introduce common spatial transformations of digital images. A spatial transformation is basically a mapping between two coordinate systems, in our cases usually between two images. So in the most general form a spatial transformation  $\mathbf{W}$  describes the relation between source coordinates  $\mathbf{x}_1$  to target coordinates  $\mathbf{x}_2$  or vice versa:

$$\mathbf{x}_2 = \mathbf{W}^F \circ \mathbf{x}_1 = \mathbf{x}_1 + (u, v)^\top \quad (2.2)$$

and

$$\mathbf{x}_1 = \mathbf{W}^B \circ \mathbf{x}_2 = \mathbf{x}_2 - (u, v)^\top \quad (2.3)$$

where  $\mathbf{W}^F$  and  $\mathbf{W}^B$  depict the forward or backward warping scheme. In a forward warping scheme each source position is associated with a target position, while in the backward warping scheme each target position is associated with its source position, Figure 3. Both approaches have several advantages and disadvantages and the choice which one to use needs to be based on the application. The benefit of the backward warping scheme is inherent prevention of unassigned data points in the warped

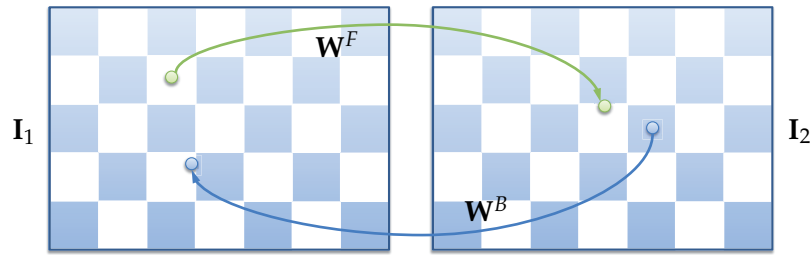


Figure 3.: Difference between forward and backward warping on a discrete lattice. In forward warping (green arrow) each pixel in the source image is associated with a position in the target image, while in a backward warping (blue arrow) each target position knows its origin in the source image.

image, i.e., for each output pixel its source position is known and can be easily queried from the source image to create the warped output. A drawback is that occlusions are hard to handle and detect. On the other hand, a forward warping scheme requires some thoughts on the image representation. As several positions in the source image might be projected to the same target position, the question arises how to combine the different samples. In addition, each source pixel, in general, influences more than a single pixel in the output image, as the warped positions are usually not discretized. Holes might also appear, as some pixels in the target image might have not been assigned by any source pixel. The two most general image representations for forward warping are therefore point-based and grid-based representations. In the point-based approach each pixel of the source image is represented as a single point and is splatted onto the target image according to its warping parameters. While being a very flexible and general representation, point-based approaches have the drawback of the aforementioned holes, and unassigned data points in the output image need to be filled. Grid-based approaches overlay a regular triangle grid on the source image and transform each vertex according to its underlying warp parameters. The image domain is still contiguous after the mapping, but one needs to deal with overlap and disocclusion that can result in visual artifacts. Both approaches can be efficiently implemented on modern programmable graphics hardware to run in real-time at almost no cost, [72, 227]. If not stated otherwise, we will use  $\mathbf{W}$  to represent the backward warping function, as it is predominantly used in this thesis, and  $\mathbf{W}_{I_1 \rightarrow I_2}$  to represent a complete pixel-dependent warp field that transforms image  $I_1$  into  $I_2$  as good as possible.



### 2.4.1 Projective Transformation

While being very general and able to represent arbitrary transformations, the aforementioned warping schemes are not always the best suited representations. An important subgroup, the *projective transformations*, rely on a mathematical formulation of the warping to represent important transformations as translations, rotations, scalings or any rigid 2D or 3D deformation. These transformations can be conveniently formulated by matrix multiplications using homogeneous coordinates. We will start with 2D transformations, the 3D equivalent can be trivially derived. A point  $\mathbf{x} = (x, y)$  in Euclidean 2-space  $\mathbb{R}^2$  is represented by a 3-tuple  $(wx, wy, w)$ ,  $w \neq 0$  in the projective plane  $\mathbb{P}^2$ . A projective transformation in this space is defined as a linear transformation of homogeneous coordinates by a non-singular matrix  $\mathbf{H}$  :

$$\mathbf{x}' = \begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \mathbf{H} \begin{pmatrix} wx \\ wy \\ w \end{pmatrix} = \mathbf{H}\mathbf{x} \quad (2.4)$$

The de-homogenization to compute the actual 2D image position of a transformed point is achieved by  $\mathbf{x}' \leftarrow (x'/w', y'/w', 1)^\top$ . An interesting property of these transformation matrices is that the multiplication is associative, i.e.

$$\mathbf{H}\mathbf{x} = (\mathbf{H}_1\mathbf{H}_2)\mathbf{x} = \mathbf{H}_1(\mathbf{H}_2\mathbf{x}) \quad (2.5)$$

and as we are dealing with homogeneous coordinates,  $\mathbf{H}$  and  $k\mathbf{H}$  describe the same transformation for all  $k \neq 0$ , therefore we will write

$$\mathbf{H} \cong k\mathbf{H}, \forall k \neq 0 \quad (2.6)$$

To categorize important transformations we group them according to the number of degrees of freedom [110]. An overview is given in Table 1.

The most specialized group of transformations is the *Euclidean group*. In the 2D case it can be represented by a  $3 \times 3$  matrix for which the upper left-hand  $2 \times 2$  matrix is a rotation matrix, the first two rows of the last column represent a translation vector and the last row is  $(0, 0, 1)$ . With this representation the motion of a rigid 2D object can be modeled. The accompanying transformation matrix with 3 degrees of freedom looks as follows:

$$\mathbf{H}_E = \begin{pmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad (2.7)$$





GROUP	DEFORMATION	INVARIANT PROPERTIES
Euclidean 3 dof		Length, area
Similarity 4 dof		Ratio of lengths, angle
Affine 6 dof		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines
Projective 8 dof		Concurrency, collinearity

Table 1.: Planar transformation hierarchy. Each row represents one group of common projective transformations. From top to bottom each group is a subgroup of the lower one and is categorized by its degrees of freedom (dof) and its most important invariant properties.

The next subgroup, called *similarity transformations*, allows for isotropic scaling in addition and is of the form

$$\mathbf{H}_S = \begin{pmatrix} k \cos \theta & -k \sin \theta & t_x \\ k \sin \theta & k \cos \theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad (2.8)$$

with  $k \neq 0$  and 4 degrees of freedom.

Fixing the last row to  $(0,0,1)$  but allowing for otherwise almost arbitrary values, always with the constraint that the re-

sulting matrix must be invertible, results in the group of *affine transformations*:

$$\mathbf{H}_A = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad (2.9)$$

The geometric interpretation of such an affine transformation can be simplified by decomposing the upper left hand  $2 \times 2$  matrix  $\mathbf{A} = [a_{ij}]$  to

$$\mathbf{A} = \mathbf{R}(\theta)\mathbf{R}(-\phi)\mathbf{S}\mathbf{R}(\phi) \quad (2.10)$$

Reading the transformations from right to left, it can be seen that  $\mathbf{R}(-\phi)\mathbf{S}\mathbf{R}(\phi)$  is simply a scaling along an arbitrary axis in the 2D plane and  $\mathbf{R}(\theta)$  is a rotation around the origin.

Allowing for the full 8 degrees of freedom results in the most general form of projective transformations, also called homographies or collinearities:

$$\mathbf{H}_P = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \quad (2.11)$$

This representation supports rotation, arbitrary scaling, translations, shearing and perspective foreshortening.

## 2.5 THE CAMERA MODEL

The previously introduced transformation model can easily be extended to more than two dimensions to model the central projection of a classic pinhole camera. In this model the image  $\mathbf{p}^i$  of a 3D point  $\mathbf{p}$  is created by calculating the intersection of a ray going from the camera's projection center  $\mathbf{C}_i$  to  $\mathbf{p}$  with the image plane of image  $\mathbf{I}_i$ , see Figure 4. We use the superscript notation  $\mathbf{p}^i$  to denote the projection of a point  $\mathbf{p}$  into the image domain of camera  $\mathbf{C}_i$ . From the intercept theorem we can derive  $\frac{y'}{f} = \frac{y}{z}$ , where the focal length  $f$  in this 2D example is the distance from the camera's origin  $\mathbf{C}_i$  to the image plane  $\mathbf{I}_i$ . In the classic pinhole model the image plane would be behind the camera, but in computer graphics it is common to place it in front of the camera to ease explanations and computations. This transformation can be conveniently described by a matrix multiplication with homogeneous coordinates:

$$\mathbf{p}^i = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \mathbf{P}_i \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} = \mathbf{P}_i \mathbf{p} \quad (2.12)$$

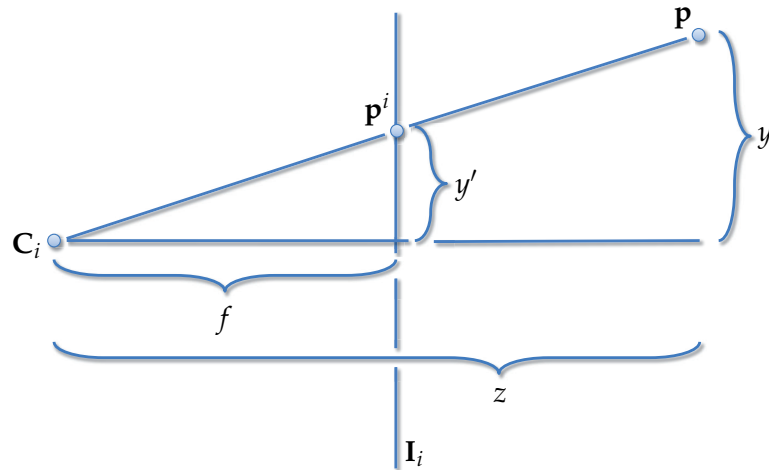


Figure 4.: Pinhole projection scheme.

where  $\mathbf{P}_i$  is a  $3 \times 4$  projection matrix with 11 degrees of freedom, basically the extension of equation (2.11) to points in  $\mathbb{P}^3$ .

One can decompose the general projection matrix  $\mathbf{P}_i$  into its extrinsic and intrinsic parameters:

$$\mathbf{P}_i = \mathbf{K}\mathbf{R}[\mathbf{I} \mid -\mathbf{C}_i] \quad (2.13)$$

Here the  $3 \times 3$  rotation matrix  $\mathbf{R}$  and the point  $\mathbf{C}_i \in \mathbb{R}^3$  describe the orientation and position of the camera in world space coordinates, and  $\mathbf{I}$  is the  $3 \times 3$  identity matrix. The  $3 \times 3$  matrix  $\mathbf{K}$  represents the intrinsic camera parameters, i.e., it defines the coordinate frame of the image:

$$\mathbf{K} = \begin{pmatrix} f_x & s & x_p \\ 0 & f_y & y_p \\ 0 & 0 & 1 \end{pmatrix} \quad (2.14)$$

$f_x, f_y$  represent the focal length, i.e., the scale along the  $x$ - and  $y$ -axis of the image coordinate frame.  $s$  is a skewing parameter, and  $x_p$  and  $y_p$  are the image coordinates of the principal point of the projection, i.e., the intersection of a line which is orthogonal to the image plane and goes through the camera's origin  $\mathbf{C}_i$ . With these parameters the projection of a 3D point into a camera is fully described. One interesting aspect, which will be heavily used in Chapters 10 and 11, is that it is possible to establish approximate correspondences between two images  $\mathbf{I}_1$  and  $\mathbf{I}_2$  given a geometric proxy  $\mathbf{G}_A$ , as one can compute the projections  $\mathbf{p}^1$  and  $\mathbf{p}^2$  of each point  $\mathbf{p}$  on the proxy in the different images.

Unfortunately, in real cameras the projection is not that simple due to lens distortion and chromatic aberration. For a correct projection these effects need to be taken into account in both projection and calibration. A variety of approaches exist to estimate

the necessary parameters, either based on images of known calibration patterns [33, 244, 280], prior knowledge of scene geometry [51, 59], or general structure-from-motion or bundle adjustment [110, 219, 243]. For the remainder of this thesis we will assume that the camera calibration is provided by one of the above-mentioned methods, and that image distortions which are not handled by the pinhole model have been taken care of in doing a preprocessing.

## 2.6 IMAGE BLENDING

*Image blending* combines two or more images to a single result by combining the weighted influences of the images. The simplest blending scheme between two or more images is therefore

$$\omega_1 \mathbf{I}_1 + \omega_2 \mathbf{I}_2 + \dots + \omega_n \mathbf{I}_n \quad (2.15)$$

with  $\omega_i \in \mathbb{R}$ . In order to keep overall intensity constant, the sum of weights is usually bound to the constraint  $\sum_{i=1}^n \omega_i = 1$ . If the blended images provide similar content at the same pixel positions this simple cross-dissolve yields high quality results. If the content differs artifacts appear, and it is necessary to adjust the different aspects of the images like color, content or resolution, as we will do in Chapter 4 of this thesis.

The above-mentioned simple weighting scheme is very restrictive as a single scalar value per image is used to provide the blending parameters. In order to provide more flexibility, e.g. spatial variation, we reformulate the weighting parameters  $\omega_i$  as functions  $\omega_i : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}$  depending on the pixel position  $(x, y)$  bound to the constraint  $\sum_{i=1}^n \omega_i(x, y) = 1$ , or even depending on the parameters of the plenoptic function  $\omega_i : \Omega \subset \mathbb{R}^5 \rightarrow \mathbb{R}$ , if this simplifies the explanation.

## 2.7 IMAGE MORPHING

Image blending provides a technique to create smooth transitions between images. But in many cases the image structures will not match. *Image morphing* combines image blending, Section 2.6, with image warping, Section 2.4, to provide a more plausible transition between two images. Image morphing dates back to the early 1980s and the experimental art by Tom Brigham [269]. It became a famous standard technique in the movie industry after its first high-quality appearance in 1988 in the Hollywood movie *Willow* and has been used for various special effects since then [23, 269].

The image morphing process between two images can be formulated as follows:

$$\mathbf{I}_{1,2}(t) = (1-t)((t\mathbf{W}_{\mathbf{I}_1 \rightarrow \mathbf{I}_2}) \circ \mathbf{I}_1) + t(((1-t)\mathbf{W}_{\mathbf{I}_2 \rightarrow \mathbf{I}_1}) \circ \mathbf{I}_2) \quad (2.16)$$

with  $t \in [0, 1]$  and  $\mathbf{I}_{1,2}(0) = \mathbf{I}_1$  and  $\mathbf{I}_{1,2}(1) = \mathbf{I}_2$ . Here  $t$  is the time parameter that influences both the color influence and amount of warping of the images. Hence, to generate a plausible intermediate image the task is twofold. The images are first warped towards each other based on the time parameter  $t$  that scales the warp fields, and then blended according to the same parameter.

## 2.8 3D RECONSTRUCTION

The warping functions described in Section 2.4 cannot only be used for image warping or morphing but also to establish 3D correspondences between two or more images, enabling one to reconstruct a complete 3D model from input images. Depending on the task only a 3D model of the foreground or a complete scene model is needed. For proper reconstruction the camera parameters need to be known in advance. These can be determined by several methods and the choice depends on the task [110, 219, 244].

As described in Section 2.5, assuming a pinhole camera model the projection of every point  $\mathbf{p}$  in a 3D scene into its image space position  $\mathbf{p}^i$  can be computed. Given this dependency between the 3D world and its 2D image equivalent, reconstruction of the scene geometry is possible if a scene point is recorded by more than a single camera. 3D reconstruction from images alone has been a vast area of research for years [63, 210, 216]. Here we will concentrate on the most commonly used and established techniques for sparse multiview setups.

### 2.8.1 Model-based Reconstruction

The Free-Viewpoint Video System of Carranza *et al.* [40] combines motion capture and 3D reconstruction by using a single template model. In a first step the silhouettes of the object of interest are extracted in all input images. A generic human body model consisting of several segments, i.e. submeshes, and a corresponding bone system is then adapted to resemble the human actor and fitted to the silhouettes of each video frame by an analysis-through-synthesis approach. A single parameterized template model cannot represent all possibilities of human shapes sufficiently, therefore the result can be improved by identifying multi-view photo-inconsistent regions and fine-tuning the mesh in these regions by enforcing a color-consistency criterion [54].

Small details usually cannot be sufficiently recovered by these methods, as the underlying mesh is quite coarse. An improvement can be achieved by acquiring a detailed mesh beforehand. Anguelov *et al.* [13] make use of detailed laser scans of an ac-

tor in different poses, from which they learn a pose deformation model and a model of variation for the body shape in order to simulate realistic muscle behavior on the model. De Aguiar *et al.* [56] also make use of detailed laser scans of the actor which they deform in order to maximize the congruence with the multi-view recordings. Their system is not aiming for realistic muscle behavior but is focused on arbitrary inputs, as e.g. humans wearing different kinds of apparel, and markerless tracking, which is less intrusive. Similar to Carranza *et al.* [40] a template model is fitted to the videos first. In a next step the laser scan is deformed to fit the template model by specifying correspondence points between the two meshes.

An even better correspondence match of the mesh with the input video can be achieved by a multi-view analysis-through-synthesis procedure, which fuses volume- and surface-based deformation schemes, and a multi-view stereo approach [57]. This allows performance captures of people wearing a variety of everyday apparel and performing energetic motions.

While this approach delivers high quality results, it is not suited for situations in which a high-quality laser scan of the actor cannot be acquired beforehand. For such situations more general methods are needed. A very interesting approach in this direction was recently proposed by Hasler *et al.* [111]. They acquired a detailed statistical model of human body shapes that describe human pose and body shape in a unified framework. Given the silhouettes of a person in several views the parameters are estimated to find the best fit of the statistical model to the given images. Although the model is based on detailed laser scans, the resulting model might only roughly fit the captured human actor. In addition, model-based reconstruction is usually performed in an offline approach. Fast model-based approaches achieving interactive reconstruction timings exist but quality suffers in these cases [61].

### 2.8.2 Shape-From-Silhouettes

The shape-from-silhouettes approach by Laurentini *et al.* [138] uses the extracted silhouettes from a finite set of viewpoints of the object to determine its approximate visual hull. In 2D the visual hull is equivalent to the convex hull, in 3D the visual hull is a subset of the convex hull possibly including hyperbolic regions. As the number of input images is limited, only an approximation of the visual hull, sometimes called inferred visual hull, can be reconstructed. It is the maximal volume constructed from backprojecting the silhouette cones of each input image into 3D space and computing their intersection, Figure 5. As this method rather conservatively estimates the real geometry,

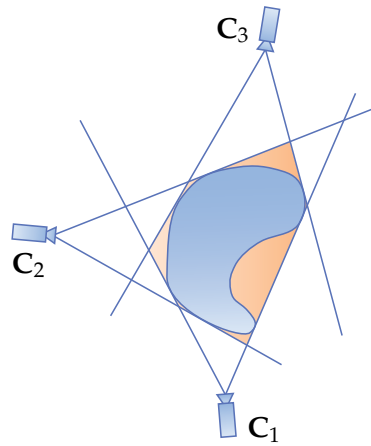


Figure 5.: The inferred visual hull (orange) of an object (blue) is estimated by reprojecting each silhouette cone and computing the intersection.

results can be quite coarse approximations of the real object. On the other hand this algorithm can easily achieve real-time frame rates [167] and can even be calculated in image-space rather than 3D space [166]. An improvement can be achieved by adding color constraints in order to detect concavities as well [133, 209] or to employ an optimization process, as it is done by Starck *et al.* [223]. Their approach combines cues from the visual hull and stereo-correspondences in an optimization framework for reconstruction, cf. Section 2.8.3.

### 2.8.3 Depth-From-Stereo

Sometimes a whole scene has to be reconstructed, in which case the previously mentioned method fail, if it is only based on silhouettes which can no longer be extracted. In this case depth-from-stereo systems perform better, as they extract a depth map for each input image, which can then be used for 3D rendering. The basic principle of depth-from-stereo is triangulation. Given two corresponding points in two images and the camera parameters, the exact position of this point in 3D can be reconstructed, Figure 6. Finding these correspondences can be arbitrarily hard and ambiguous. To relax the problem of doing an exhaustive search for similarity over the whole image, one usually makes use of the epipolar constraint to reduce the search to a 1D line search along the epipolar lines, Figure 6. Usually a rectification precedes the line search so that it can be performed along the same scanline, i.e. the input images are projected onto a plane parallel to the baseline between the optical centers of the input cameras [87]. For improved robustness, correspondence finding can be performed, for example, by window-based cross corre-



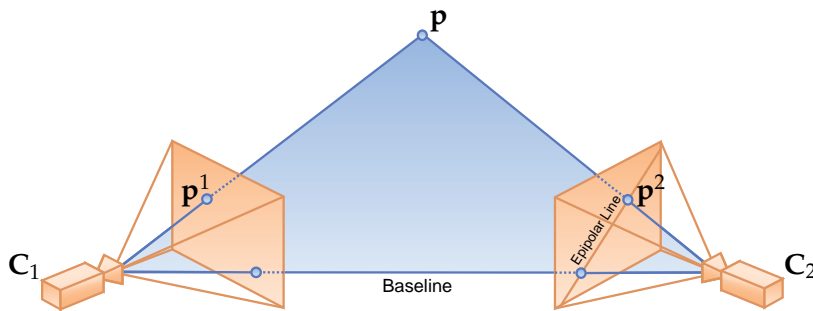


Figure 6.: Using epipolar constraints and triangulation the 3D position of any static scene point visible in both views can be reconstructed.

lation [109]. If further knowledge about the scene is given or scene constraining characteristics are assumed, as for example local smoothness, more sophisticated methods based on energy minimization can be employed [27, 32]. If more than two images can be used for depth estimation plane sweep algorithms perform well [50]. In this approach a plane is placed at different depths. The input images are projected onto it, and the plane is rendered from the virtual viewpoint. The color variation at every fragment serves as a quality estimate for this depth value. This approach is especially appealing in real-time acquisition systems, as it can be computed very efficiently on graphics hardware [89, 147, 276]. Even dedicated hardware is nowadays available for multi-view stereo reconstruction and has already been successfully applied in an image-based rendering system [177].

One of the first systems to achieve high quality interpolation with a relatively sparse camera setup was the approach by Zitnick *et al.* [281]. Instead of matching single pixels or windows of pixels, they match segments of similar color. As they assume that all pixels inside a segment have similar disparities, an over-segmentation of the image is needed. The segments are then matched and the estimated disparities are further smoothed to remove outliers and to create smooth interpolations between connected segments belonging to the same object.

Methods based on this matching approach are commonly used only for dense stereo, i.e. the distance between cameras and resulting disparity is rather small. For larger distances, or fewer cameras, additional information is needed for reconstruction. Waschbüsch *et al.* [258] use video bricks which consist of a color camera for texture acquisition and two calibrated grayscale cameras that are used together with a projector to estimate depth in the scene using structured light. The benefit of these bricks is that depth ambiguities are resolved in textureless areas. These depth estimations are used as initialization for geometry filter-

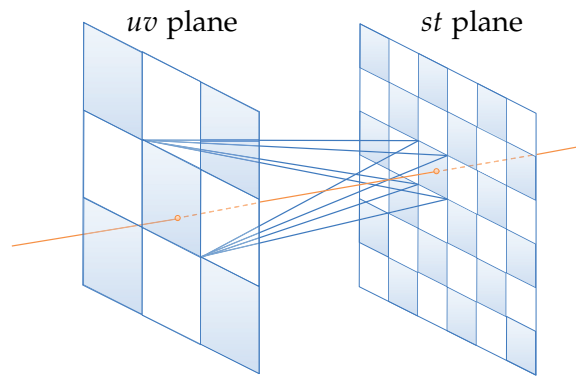


Figure 7.: Light Field Rendering: In a two-plane parameterized light field the information to reconstruct the plenoptic function is resampled into a 4D ray space [146]. The  $uv$  and  $st$  plane represent the camera plane and the focal plane, respectively. Any novel ray (orange line) is then interpolated from nearby samples (blue lines) in this representation. For clarity only a few samples are shown.

ing, based on bilateral filtering, to generate time-coherent models, removing quantization noise and calibration errors.

A recent comparison of some more multi-view stereo reconstruction algorithms can be found in [210]. There are many other 3D reconstruction methods, e.g. Shape-from-Texture [29] or Shape-from-Shading [62]. But these are commonly not used for multi-view stereo reconstruction and therefore we refer the interested reader to the appropriate literature.

## 2.9 FREE VIEWPOINT VIDEO

In classic movie making the director needs to decide beforehand how the camera moves through and records the scene. The goal of free-viewpoint video is to provide the possibility to move freely around in a scene after it has been recorded [40, 57, 223, 281]. What is needed for this additional degree of freedom is a precise reconstruction of the plenoptic function, Section 2.2. Generally, there is a continuum of possibilities to achieve this goal. On the one end we have purely image-based approaches, like the *light field* [146]. A large amount of images, plus a few restrictions to project the 5D simplified plenoptic function into a 4D ray space representation, allows for almost direct sampling and reconstruction of the target image for arbitrary viewpoints, Figure 7. On the other end of the continuum, geometry-based approaches try to deal with missing information in the plenoptic function by providing detailed geometry proxies that represent the captured scene, cf. Section 2.8. These proxies can be used to establish correspondences between the input views and the virtual camera, as described in Section 2.5.

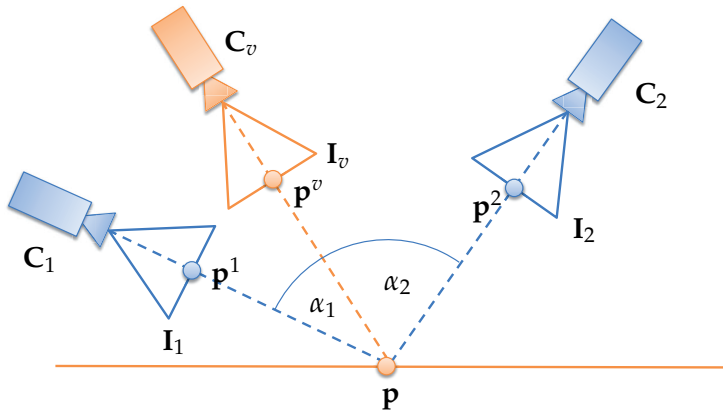


Figure 8.: Classic weighting scheme of input views based on the angular deviation. The influence of camera  $C_1$  for the depicted viewing ray should be weighted higher than the influence of  $C_2$ , as the angle between the viewing rays observing scene point  $\mathbf{p}$  is smaller between  $C_1$  and  $C_v$  than between  $C_2$  and  $C_v$ .

Techniques for new view synthesis render novel output views based on the original content of the input images [59]. Thus, for each pixel  $\mathbf{p}^v$  in the output view  $I_v$ , one has to determine the color contribution of all relevant input views in which the scene point  $\mathbf{p}$  is visible. For instance, given the two input views  $I_1$  and  $I_2$  in Figure 8, the color of pixel  $\mathbf{p}^1$  projected onto the surface and reprojected into  $I_v$  should be weighted stronger than the color of pixel  $\mathbf{p}^2$  for producing the output color of  $\mathbf{p}^v$ , since  $\alpha_1 < \alpha_2$ . I.e., the angle between the viewing rays passing through  $\mathbf{p}$  is smaller for camera  $C_1$  and the virtual camera  $C_v$ . In general, these color contributions can be computed based on blending weights  $\omega_i$  with

$$\mathbf{I}_v(\mathbf{p}^v) = \frac{1}{\sum_i \omega_i(\mathbf{p}^v)} \sum_i \omega_i(\mathbf{p}^v) \mathbf{I}_i(\mathbf{p}^i) \quad (2.17)$$

This projection technique is also called multiview projective texture mapping or view-dependent texture-mapping [59]. To reduce visual artifacts in this simple blending scheme, several aspects like viewing angle, visibility, spatial and temporal continuity, can be integrated in the computation of reasonable weights, as investigated by Buehler *et al.* [36]. These simple weighting schemes, which are basically projected image blending as described in Section 2.6, give correct results if certain conditions are fulfilled, like correct camera calibration and a very precise geometry representation of the scene. In addition, non-diffuse materials can only be approximated.

All of the above-mentioned constraints are hard to fulfill in practical applications. Acquisition with more than a few cameras is very costly and not affordable for everyone. Precise 3D reconstruction is not always possible without additional hardware,

like laser scanners [55, 56, 57] or special cameras [258]. Real-time applicability, e.g. for the transmission of live sports events [115], poses additional requirements on the reconstruction, resulting in even less robust results.

In Chapter 10 and 11 we will investigate how to loosen some of these constraints. Our work in these chapters aims at high-quality free-viewpoint video with only sparse camera setups, Figure 9, imprecise camera calibration, and approximate geometry.

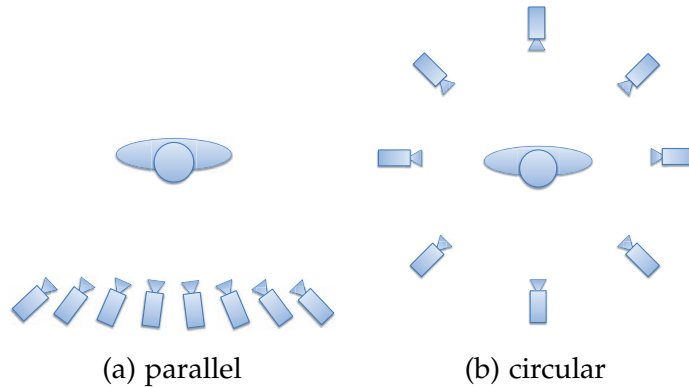


Figure 9.: Classic camera arrangements for free-viewpoint video with sparse camera setups.

## 2.10 OPTICAL FLOW

*Optical flow* estimation has a long-standing history especially in the field of computer vision [116, 159] and is frequently used for dense motion estimation between images. The assumption made is that the scene flow, i.e., the real 3D motion in a scene, can be approximated by the apparent motion in the images. It should be noted that the apparent motion might differ from the projected scene flow, i.e., the projection of the true 3D motion of an object onto the image plane. The warping formulation  $\mathbf{W}_{I_1 \rightarrow I_2}$  introduced in Section 2.4 is related to optical flow in that it is based on per-pixel motion between different images.

Optical flow estimation is generally based on the so-called *brightness constancy assumption* assuming that the intensity of a moving object does not change from one image to the next and brightness changes are only due to motion. Therefore the intensity value at all corresponding pixels in the images  $I_1$  and  $I_2$  should be approximately the same:

$$I_1(x, y) - I_2(x + u, y + v) \approx 0 \quad (2.18)$$

This formulation is susceptible to linear changes in the brightness. Therefore, the *gradient constancy assumption* is added, assuming the gradient is approximately invariant under motion:

$$\nabla \mathbf{I}_1(x, y) - \nabla \mathbf{I}_2(x + u, y + v) \approx 0 \quad (2.19)$$

$\nabla \mathbf{I}(x, y)$  is the image gradient at position  $(x, y)$ .

The solution to equation (2.18) and (2.19) can be ambiguous. Essentially only one linear equation for the two unknown motion components is given. To solve this underconstrained system additional assumptions are necessary. A common approach is to impose a regularization on the motion field, resulting in a piecewise smooth flow field where neighboring pixels should have similar motion vectors. Hence

$$\nabla u(x, y) \approx \vec{0} \quad \nabla v(x, y) \approx \vec{0} \quad (2.20)$$

The actual energy formulation that is to be minimized based on these assumptions and the according algorithm is subject to a vast number of research activities [19]. E.g. our GPU optical flow used in Chapter 11 uses the following energy formulation, which is based on the work of Brox *et al.* [35]:

$$\begin{aligned} E(u, v) &= E_{Data}(u, v) + \alpha E_{Smoothness}(u, v) \quad (2.21) \\ E_{Data}(u, v) &= \int_{\Omega} \psi(|\mathbf{I}_1(x, y) - \mathbf{I}_2(x + u, y + v)|^2 \\ &\quad + \gamma |\nabla \mathbf{I}_1(x, y) - \nabla \mathbf{I}_2(x + u, y + v)|^2) dx dy \\ E_{Smoothness}(u, v) &= \int_{\Omega} \psi(|\nabla u(x, y)|^2 + |\nabla v(x, y)|^2) dx dy \end{aligned}$$

The function  $\psi(s^2) = \sqrt{s^2 + \epsilon^2}$  with  $\epsilon > 0$  is used to achieve a robust energy function, which reduces the influence of outliers.  $\alpha$  and  $\gamma$  are weighting parameters for the smoothness of the result and for the influence of the gradient constancy assumption, respectively.

A common technique to speed up the optical flow computation and to also allow for larger displacements is to use a multiscale approach [11]. The optical flow is then computed in a coarse-to-fine fashion, i.e., the solution for the coarsest level of an image pyramid is evaluated and the solution is then upsampled and used as the initialization for the next level until the final resolution is reached.

There are a lot more assumptions that can be incorporated in the energy formulation of an optical flow algorithm, like color-spaces, different regularizers or optimization strategies [19, 225, 279].

## 2.11 MATTING

The term *matting* refers to the problem of accurate foreground estimation in a single image or video sequence. The goal is

to extract an object from image footage and create novel composites, which is a very common editing task in movie productions. For many complex objects, such as hair or fur, it is important to not only extract a binary matte but to determine both full and partial pixel coverage, also known as *pulling a matte*, *alpha matting* or *digital matting* [253]. Since its early days back in 1984 when Porter and Duff established the mathematical problem [189], there have been intensive research activities to find automatic or semi-automatic approaches to extract the foreground. Mathematically, the *compositing equation* describes the digital matting process as a linear combination of foreground color  $\mathbf{I}_F$  and background color  $\mathbf{I}_B$  in every pixel of an image  $\mathbf{I}$ :

$$\mathbf{I}(x, y) = \alpha(x, y)\mathbf{I}_F(x, y) + (1 - \alpha(x, y))\mathbf{I}_B(x, y) \quad (2.22)$$

where  $\alpha(x, y) \in [0, 1]$ . If  $\alpha(x, y) = 0$  the pixel at position  $(x, y)$  is called definite background and if  $\alpha(x, y) = 1$  definite foreground. In most natural images, most pixels are definite foreground or background. For accurate foreground estimation, it is important to estimate the alpha values for mixed pixels as well. The problem is severely ill-posed, as there are seven unknowns, two *rgb*-triples and one  $\alpha$ -value, and only one known color vector per pixel. Therefore, additional information such as user-constraints or prior assumptions on image statistics need to be incorporated [17, 253, 255].

## 2.12 GRADIENT DOMAIN COMPOSITING

In a number of problems in computer graphics and vision energy functions need to be minimized which are defined in the form of a large linear system of equations:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (2.23)$$

In such systems the solution vector  $\mathbf{x}$  is the unknown solution, while  $\mathbf{A}$  and  $\mathbf{b}$  are given. While it might be unconventional from a graphics perspective to think in terms of linear systems of equations, this representation has several benefits for image editing and optimization tasks. By vectorizing an image, i.e. concatenating each row of a grayscale image to get a 1D representation a lot of classic image editing tasks can be represented. For example, the Laplacian  $\nabla^2$  of an image can be computed by the  $n \times n$  matrix

$$\begin{pmatrix} & & & & & & & \ddots & & & & & & & \\ 0 & \dots & 1 & 0 & \dots & 1 & -4 & 1 & 0 & \dots & 1 & 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 & 0 & \dots & 1 & -4 & 1 & 0 & \dots & 1 & 0 & \dots & 0 \\ & & & & & & & \ddots & & & & & & & \end{pmatrix},$$

(2.24)

where  $n$  is the number of pixels. In this case  $\mathbf{x}$  would be the given image and  $\mathbf{b}$  is the solution vector, so the solution is straight forward to compute. Of course, in most cases this representation for image editing tasks is only interesting from a theoretical point of view, especially because setting up the full matrix  $\mathbf{A}$  requires  $n^2$  entries to be saved, which is prohibitive for images of the size of several million pixels.

Still, there are two interesting findings which make this approach suitable for image editing tasks. First, most image editing operators can be represented in a very sparse matrix  $\mathbf{A}$ , so the matrix does not need to be saved explicitly. And given this representation, one can actually solve the inverse problem. For example, given the gradient images and the gradient operator, what does the original image look like? Though it should be noted that for this example the solution is only defined up to an additive constant. This is also the idea behind the *gradient domain compositing* in image editing tasks such as seamless cloning [5, 126, 184], panorama stitching [4, 143, 215], inpainting [264] and many others. The goal is generally to compose two images by adjusting the colors of (at least) one of them to create a seamless composite.

Given a source image  $\mathbf{I}_S$  and a target image  $\mathbf{I}_T$  as well as a region of interest  $\Omega$  with boundary  $\partial\Omega$ , one seeks to find an optimal solution  $\mathbf{I}_R$  in a least-squares sense to the following minimization problem:

$$\min_{\mathbf{I}_R} \int_{\Omega} |\nabla \mathbf{I}_R - \nabla \mathbf{I}_S|^2 \text{ with } \mathbf{I}_R|_{\partial\Omega} = \mathbf{I}_T|_{\partial\Omega} , \quad (2.25)$$

where  $\mathbf{I}|_{\partial\Omega}$  denotes the pixels on the seam  $\partial\Omega$  and  $\nabla$  is the gradient operator. One searches for the best fitting image that fulfills two constraints: The boundary pixels  $\mathbf{I}_R|_{\partial\Omega}$  must match exactly the colors of the target image  $\mathbf{I}_T|_{\partial\Omega}$  and for the interior  $\Omega$  the gradient should match as good as possible the gradient of the source image  $\mathbf{I}_S$ . Figure 10 illustrates the notations. The unique

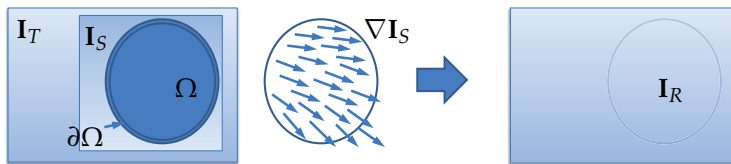


Figure 10.: Gradient domain compositing. Given the image functions  $\mathbf{I}_S$  and  $\mathbf{I}_T$  as well as a composite region  $\Omega$ , one seeks to find the best interpolant  $\mathbf{I}_R$  in  $\Omega$  under the guidance of  $\nabla \mathbf{I}_S$  and given boundary values along  $\partial\Omega$  so that  $\mathbf{I}_S$  can be seamlessly composited into  $\mathbf{I}_T$  in the region  $\Omega$ .

solution to the minimization problem in (2.25) can be found by

solving the *Poisson equation* with Dirichlet boundary conditions [184]:

$$\nabla^2 \mathbf{I}_R - \mathbf{div}(\nabla \mathbf{I}_S) = 0 \text{ with } \mathbf{I}_R|_{\partial\Omega} = \mathbf{I}_T|_{\partial\Omega} \quad (2.26)$$

which equals

$$\nabla^2 \mathbf{I}_R - \nabla^2 \mathbf{I}_S = 0 \text{ with } \mathbf{I}_R|_{\partial\Omega} = \mathbf{I}_T|_{\partial\Omega} \quad (2.27)$$

where  $\mathbf{div}$  is the divergence operator  $\frac{\partial}{\partial x} + \frac{\partial}{\partial y}$  and  $\nabla^2$  is the Laplacian operator. In the slightly more general form of the Poisson equation,  $\nabla \mathbf{I}_S$  can be any vector guidance field  $\mathbf{v} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ . The Poisson equation is defined only for scalar functions. For color images Equation (2.26) must be solved for each color channel separately.

If one takes a closer look at the given constraints for each pixel of the resulting function, it turns out that: Either a constraint scalar value is given for a pixel, or the Laplacian should equal a certain value. The problem in Equation (2.26) can then be reformulated as a linear system of equations of the form  $\mathbf{Ax} = \mathbf{b}$ . The matrix  $\mathbf{A}$  looks similar to the one given in (2.24) except that a color constraint for the  $i$ -th pixel can be set by exchanging the  $i$ -th row of  $\mathbf{A}$ , with a row consisting of only zeros except for the  $i$ -th entry, which is set to 1. The vector  $\mathbf{b}$  contains the intensity values of  $\mathbf{I}_T$  for each entry belonging to  $\partial\Omega$ , and the value of the Laplace operator applied to  $\mathbf{I}_S$  for the rest. This huge linear system can then be solved with existing linear solvers [190]. We will make use of gradient domain compositing in Chapter 4 to adjust the color of input images.

### 2.13 EXEMPLAR-BASED TEXTURE SYNTHESIS

Texturing is a core process for providing details beyond geometric resolution. There are many different ways to acquire a texture [137, 176, 250], but exemplar-based texture synthesis provides one of the most flexible and interesting methods [261]. The idea behind exemplar-based texture synthesis is to provide the algorithm with a small exemplar patch of a specific texture from which an arbitrarily large output texture is created, which is not only visually similar, but does not contain any unnatural artifacts or repetitions.

Since the seminal work of Efros *et al.* [65], a lot of different approaches have been proposed, which can be divided into three categories: Pixel-based synthesis, patch-based synthesis, and texture optimization. Most methods model a texture as a realization of a local and stationary process based on Markov Random Fields [261]. The assumption is that under a proper window size, the observable portion of the texture always appears similar, and



each pixel is predictable from a small set of neighboring pixels and is independent of the rest of the image.

In pixel-based synthesis methods this model is integrated by synthesizing the image one pixel at a time. Each output pixel is determined by a neighborhood search process. Using the already determined pixels around a new pixel that is to be synthesized, the algorithm searches for a similar region in the input image to replace this pixel [65, 260].

The computationally intensive neighborhood search can be accelerated by nearest neighbor search techniques, like kd-trees or tree-structured vector quantization [260]. Other effective methods are based on the notion of coherence. The k-coherence algorithm by Tong *et al.* [242] precomputes for each neighborhood around the input pixels the best matching neighborhoods in the same exemplar. During synthesis it is very unlikely that pixels from the input will land on random output locations; instead, they have a tendency to stay together also in the output, which makes k-coherence algorithms quite effective.

Patch-based synthesis algorithms do not copy single pixels but rather whole patches from the input to the output image. A benefit in terms of speed and quality can be obtained because fewer patches than pixels are needed and only the transition areas between the patches need to be adjusted to create a high-quality output. To hide the transition, patch-based approaches try to partition the overlapping areas by using dynamic programming [64] or graph cut optimization [134].

The most effective approaches so far are texture optimizations [108, 135] which combine properties of both pixel-based and patch-based algorithms. Unlike pixel-based approaches, texture optimization does not synthesize pixels one by one but considers them altogether and optimizes the pixels with respect to a quadratic energy functions which is determined by mismatches of input/output neighborhoods. We will make use of the theory behind texture optimization to hallucinate details to our output image in Chapter 4.



Part II.

Error Concealment in  
Seamless Image  
Synthesis



*A computer is fast and accurate, but it also is completely literal.  
It doesn't know enough to correct your simplest mistakes;  
it takes everything you enter exactly as you entered it, and not as you  
meant it!*

— Peter Aitken

### 3.1 BACKGROUND

The goal of image-based rendering is to create a visually plausible and convincing sense of presence in a scene using only photographs or videos. In the early days of digital photography this sense was often limited due to hardware restrictions of the image resolution. Panorama imaging has become a convenient way to manually increase the resolution by stitching several images together [236]. These programs are welcomed extras to most digital cameras, there are even apps available for mobile phones [1]. Nowadays, even gigapixel images have become a more and more common and well-liked amusement [131]. Unfortunately the generation of such a large image comes with several problems. Artists like Graham Flint have to use specialized cameras to capture these high-resolution images [83]. Others, like the gigapixel project by Kopf *et al.* [131] rely on a carefully roboter-controlled camera mount and panorama stitching techniques to create an image several times larger than the original resolution of the camera.

A common drawback of all panorama techniques is that they are, in general, not able to deal with missing information. Insufficient input images will lead to blur, in the best case, or completely empty regions in the stitched panorama result, in the worst case. At the same time the internet allows for photo sharing at a massive scale today. For example, the phrase "Big Ben" returns more than 242,000 images on Flickr [274]. Can image databases be used to fill in missing details in images?

Not only the creation but also the display of large images poses problems. For very large virtual images, e.g. the gigapixel panoramas created by the 360 Cities community [47], Google Earth [98], etc., streaming technologies are a necessity, loading the needed image data on demand, although with a noticeable time lack. There is another area where the possibility to dis-

play large images is a welcomed feature but where such a lack would not be tolerable: computer games. Here images are used to texture the different surfaces in the scene. If one takes a look back at the history of games it becomes obvious that the amount of representable realism and texture sizes used to go almost hand in hand. The drawback of streaming technologies here is that the eye movement is almost not foreseeable, resulting in less caching efficiency as the needed data cannot be loaded fast enough into the graphics card memory. In this part of the thesis we propose solutions to both mentioned problems: the creation of high-resolution image content from incomplete, low-resolution images, in Chapter 4, and a rendering technique for sparse texture maps, where only the detail needed is additionally stored, reducing the need for texture streaming, Chapter 5.

### 3.2 RELATED WORK

#### PANORAMAS, GIGAPIXEL IMAGES AND PHOTO BROWSING

Capturing and creating panoramic images is an idea almost as old as photography itself. A very nice survey on related techniques can be found in [236]. Most approaches assume that the camera stays roughly in the same place during acquisition. With a carefully designed camera setup, Kopf *et al.* [131] take hundreds of images to produce one Gigapixel image by stitching the photos together. The approach creates beautiful results that can be explored interactively, but the setup is complex and requires special hardware. In comparison, our system, presented in Chapter 4, creates high-resolution images from unordered sets of input images. Only a few restrictions apply to our input images, making acquisition easy. Even image databases can be used.

Building upon the work by Snavely *et al.* [219, 220], Microsoft Research recently released their Photosynth Tool [198]. It provides a special 3D interface that allows the user to easily navigate a large photo collection of a particular location. While the interface resembles a 3D environment and is quite intuitive to use, color correction and blending between photos was only added for convenience and still reveals many artifacts. Our approach addresses these points to produce a high-quality output where different photos are merged in a common 2D domain. It works without any 3D information or camera calibration, but instead computes dependency relations in order to faithfully transfer details between different shots.

Recently, another interesting system for exploring large collections of photos in a virtual 3D space has been presented by Sivic *et al.* [215]. It allows virtual walkthroughs of a photo collection by stitching similar scenes into one browsable image graph,

similar to a large 2D image, but transitions are still visible. In contrast, we address this problem to create seamless transitions between the input images at various scales, but reject images not belonging to the same scene.

#### IMAGE STITCHING

A very nice survey on image stitching can be found in [236]. Local approaches blend the colors of overlapping images based on precomputed weighting masks [235, 248]. The multiresolution spline by Burt and Adelson [37] adjusts the transition separately for each band of frequencies, based on a Laplacian pyramid to prevent ghosting artifacts and sudden transitions between the images. Instead of blending between the images it can be more favourable to select only one input image per output pixel in the overlapping area to prevent artifacts such as ghosting. This choice is usually based on an optimal seam between the images. Taking the color difference in the overlapping area into account, an optimal partition based on Graph Cuts [32] or dynamic programming [24] is usually computed. The image stitching algorithm by Levin *et al.* [143] operates directly in the gradient domain to compute an optimal path based on the gradient strength. Optimal seam methods usually require a good alignment of image features beforehand in order to reveal pleasing results. This, however, cannot be guaranteed in most image stitching applications and misaligned edges or misplaced features are the result.

Han *et al.* [106] create a visually smooth image pyramid from already stitched imagery at several scales to hide jarring transitions when zooming into the images, e.g. in applications like Google Earth [98]. They combine the detail of one image with the local appearance of another and use clipped Laplacian blending to minimize blur for the intermediate levels in the pyramid which need to be created in addition. This way color and structure are conserved, but the method requires images with similar content at different levels of detail in order to produce plausible results.

Structure deformation for image alignment has been heavily researched in medical image registration, which commonly deals with non-rigid registration errors by first roughly aligning the images (if this is not inherently done), matching prominent features and smoothly interpolating these sparse correspondences to compute a global deformation field. This approach of constraining the deformation and enforcing interior smoothness was first proposed by Bajcsy and Kovacic [18]. An overview of the large amount of existing literature for medical image registrations is given in [162]. Matching features and distorting the image accordingly has also been used in a variety of applications as texture synthesis or especially image morphing. Wu and Yu's

texture synthesis algorithm in [272] deforms texture patches by matching sparse features and interpolating the deformation vectors based on thin-plate splines [172] or Shepard’s method [118]. But no intensity correction is applied.

The work with the closest resemblance to our structural adaptation presented in Section 4.3.2 is the approach by Jia *et al.* [124, 125]. They aim at correcting the mismatch along the partitioning border between two images by feature matching. The computed deformation along the seam is then smoothed out into the interior of the source, without taking any further structural information into account. Our approach aims at overcoming these limitations, by not only matching features along the seam, but also by tracing salient edges into the images, which are used as additional deformation constraints to create more plausible transitions.

#### SUPER-RESOLUTION

Super-resolution is a heavily researched area with various instances of algorithms based on exemplar-images or learning-based methods [249]. One approach is to derive image statistics from the image itself or a database of images [43, 92, 114, 230, 233, 275]. Other approaches rely on edges, gradients, or combinations with learning-based methods [53, 78, 85, 232], reconstructed 3D geometry [27] or specialized hardware [104]. Similar in spirit to our method in Chapter 4 is the method by Ancuti *et al.* [12] to upsample a low-quality video based on a few high-resolution detail photographs. It works well if a region is covered in detail shots, but shows weaknesses if such information is missing. Furthermore, hierarchical dependencies between images are not being considered, limiting the upsampling capability of classic super-resolution approaches.

Despite good quality at moderate magnification of the images, super-resolution approaches are usually far from real-time capable and are not applicable to high magnification factors as the results will start to look flat and non-photorealistic. An additional drawback is the inherent problem of super-resolution itself: super-resolution is bound to the constraint that the down-sampled image must look exactly like the original, which can be a drawback if the image is noisy or contains quality flaws like compression artifacts. Instead, our approach, in Chapter 4, only assures similarity to the original images, therefore giving the algorithm more possibilities to create plausible, new details in the upsampled image.

#### EXEMPLAR-BASED TEXTURE SYNTHESIS

Exemplar-based Texture Synthesis approaches create larger texture maps from one or more small exemplar patches. One well-



known approach is the image quilting technique by Efros and Freeman [64], in which a new image is synthesized by stitching together small patches of existing images. Kwatra *et al.* [134] build upon this approach by using a graph cut technique to determine the optimal patch region to be used for synthesis. Constrained texture synthesis tries to guide the texture creation process [108, 139, 194, 261]. The usual approach is to take neighbor information of a pixel into account and to minimize some cost function which varies from approach to approach.

For faster generation, tile-based approaches can be used [49, 259]. While the creation of periodic texture tiles is relatively simple, the periodicity can be annoyingly apparent for certain textures. Wang tiling can be used to allay this effect by creating patches, called Wang Tiles, which can be arranged together to non-periodically tile the plane.

All these approaches only synthesize textures at a specific scale, i.e., features are usually not enlarged or shrunk in any way. In contrast, Ismert *et al.* [122] add detail to undersampled regions in an image-based rendering setup if more-detailed versions of the same texture are available in the same image. Wang and Mueller [256] present an approach where a low-resolution image guides the texture creation process for the higher resolution details. Only recently, Han *et al.* [107] have presented an approach that uses patches at different scales for the synthesis process.

The problem with any of these texture synthesis approaches is that they are only suitable for textures with relatively similar repeating structures, although non-periodically arranged.

We will make use of the idea behind exemplar-based texture synthesis, in Chapter 4, to transfer details from different input images to our upsampled output image result. If regions are not covered by any input image, we fill in the missing parts with plausible details from the other images at the appropriate scale.

#### TEXTURE MAPPING

Creating or capturing high resolution images is only one part of texture mapping with high-resolution imagery, as efficient means to display these images are also needed. A common approach is to represent images as textures and map them onto some geometry proxy to render them using the standard graphics pipeline. Texture mapping was introduced in computer graphics in 1974 [41] as a very efficient way to increase visual rendering complexity without the need to increase geometric detail. To overcome the aliasing problems apparent when the texel-to-pixel ratio exceeds unity, also known as minification, Williams introduced the mipmap representation [268], a pre-calculated image pyramid at different resolutions of the texture. Advanced varia-

tions, like ripmaps [8] or mipmaps [31], solve this problem with even higher quality, but at the cost of higher memory requirements or slower rendering. Other possibilities are summed area tables [52] or elliptical weighted average filters [101]. A survey of classic texture mapping can be found in [113]. Usually this minification is only addressed for a single texture map. If textures overlap or texture insets are created, flickering artifacts can appear and borders might become visible. We will address this problem in Chapter 5.

While the problem of single texture minification is well solved, the problem of texture magnification is still a very active area of research.

#### TEXTURE INTERPOLATION

The most common approach in texture interpolation, which is still used in most computer games due to its simplicity and availability in standard APIs like OpenGL or DirectX, is to select either the nearest-neighbor or to linearly interpolate color values between neighboring texels. Using nearest-neighbor results in blocky artifacts, while linear interpolation gives blurry results.

In many applications, such as games, the texture is represented by a texture atlas. Interpolation during texture lookup provides a continuous value field everywhere on the surface, except at the chart boundaries where visible discontinuities appear. This problem is addressed in the work by Ray *et al.* [195] who use a quad remeshing technique to reparameterize the texture atlas in order to hide the seams. Therefore their work is related to our approach presented in Chapter 4 and 5, as we merge different textures to hide the seams between them.

#### LARGE TEXTURES

The most straight-forward idea for providing detail in textures is to simply use sufficiently large textures which are dynamically loaded on demand. But memory as well as bandwidth limitations restrict textures to a maximum size. Tanner *et al.* [238] address this problem by introducing clipmaps. In this approach the texture data is loaded on demand depending on the viewer's position. This approach works particularly well for mapping height fields [120, 157], as needed e.g. in geographic information systems (GIS).

In all these approaches only scenes are considered where the needed data is in direct relation to the current viewpoint. This makes texture prefetching possible because the needed data does not change abruptly. However, this is not always the case. In general texture mapping applications the rendered scene might be dynamic, or the viewpoint might change abruptly, e.g. if the user turns around quickly. Therefore, representations that reduce the

memory amount are necessary to reduce the need for texture streaming.

#### VECTOR TEXTURES

Texture maps are usually represented as a collection of discrete image elements and are therefore always limited in spatial frequency. Instead of using samples taken from the underlying texture function, vector textures represent a continuous function using resolution-independent primitives. Tumblin and Choudhury [245] save sharp boundary conditions at discrete positions for every texel to prevent some of the strong blurring apparent in usual texture magnification. Sen [212] uses silhouette maps to maintain sharp edges in the texture while blurring at smooth transitions. A similar approach by Tarini and Signoni [239], called pinchmaps, extends this approach to curved primitives. A complete support for all primitives of a SVG description in a vector texture was presented by Qin *et al.* [193], building on their own previous work in [192]. Recently, Jeschke *et al.* [123] showed how to render surface details using diffusion curves onto arbitrary meshes.

The drawback of vector textures is that they can only preserve low frequency components, overall color, and very high frequency components, the strong edges, while mid-frequencies and new details are not present in a close-up view. This can give vector textures a cartoonish and unnatural look. In a sense, this problem is related to the problem encountered in most super-resolution approaches.

#### MULTIRESOLUTION TEXTURES

Multiresolution and multiscale textures represent textures by using a hierarchical representation. They most resemble our work presented in Chapter 5. In the early days hierarchical texture representations were mostly used for multiresolution paint programs [26, 185] where wavelet or bandpass representations are used in a quadtree structure created on demand.

A first approach to emulate high resolution textures in real-time rendering was to use detail textures [30]. The idea is to save a tile of typical high-frequency information in a separate texture, which is then drawn on top of the usual texture during rendering. In modern computer games, the use of virtual textures has become widespread [174]. Here, a high-resolution texture is subdivided into equal-sized tiles, which are saved on the hard disk drive. An indirection texture saves the indices of the tiles. During rendering the indirection table is used to query the according texture value from the tiles, which are streamed onto the GPU on demand. This approach is fairly efficient as only two texture lookups are needed in the simplest case, plus,

it can be extended to support mipmapping and also sparse representations [21]. A minor drawback is the still-limited resolution, restricted by the size of the indirection table and size of the tiles. In general, this is enough for many applications, but would not be enough for some of the examples we are showing in Chapter 5. Furthermore, dynamic updates are difficult to integrate.

Finkelstein *et al.* [80] use binary trees of quadtrees to encode multiresolution video. Ofek *et al.* [180, 181] and Mayer *et al.* [169] create a quadtree texture from a series of photographs. But quadtree structures might not be the best representation for texture maps, as it may take up to  $\log(n)$  texture lookups for a texture fetch depending on implementation. Also, filtering can become more difficult as neighboring texels might not be available. In contrast, our approaches can make use of the built-in hardware texture filtering functionality of the GPU.

Kraus and Ertl [132] divide an already given high-resolution image (or 3D or even 4D volume) into a regular grid of fixed-sized blocks. The information residing in these blocks is resampled into a common texture map, reducing the size of blocks in regions with low detail information. The grid then serves as an indirection table into the actual data during rendering. Using the same texture for all patches may, however, result in problems when applying mipmapping, plus, indirection tables are inefficient to update if the texture should change over time. Lefebvre and Hoppe [140] use a compressed adaptive tree structure which allows for fast random access on current graphics hardware while achieving large reductions in memory requirements. However, the requirement for large input textures to the algorithms remains as a drawback.

Lefebvre *et al.* [141] also presented an interactive approach to add small texture elements, called texture sprites, onto an arbitrary surface. Basically, their structure resembles an octree where the texture sprites are saved at the leaf nodes. Their implementation is very memory-efficient and allows for various artistic effects, but is less suited for hierarchical texture representations.

To overcome the need of explicit parameterization, Benson and Davis [25] introduced octree textures. Using an octree instead of a quadtree allows for encoding the spatial relationship directly in the position in the octree. It also overcomes the problem of wasted texture space usually encountered in classic 2D texture atlases [60, 136]. The drawback is high access cost, since for each texture lookup one needs to traverse the octree from the root to the leaf node.

*We don't want to go into too much detail here...*

— Various authors

#### 4.1 INTRODUCTION

Applications like Photo Tourism [219] or Microsofts Photosynth [198], Google Earth with Street View [98] as well as other projects show that there exists an ever increasing interest in finding new ways to deal with photo collections, as image acquisition becomes easier and cheaper. Our goal in this chapter is to analyze the difficulties encountered in seamless image compositing of multiple images. For this purpose we developed an application that relies on multiple photos to add high-resolution details to a chosen input photo. The user can improve a holiday snapshot so that it becomes possible to zoom in and take a closer look at interesting parts of the image, far beyond the original image resolution. Starting with an unordered collection of arbitrary images, our system automatically arranges them in a dependency graph that describes which photograph contains details of another photograph. The user then chooses a photo, and the system seamlessly enhances it up to the desired resolution with details found in other images.

Resolution enhancement is of particular interest when the images are shown on larger screens where an insufficient resolution is most visible. But it also receives much interest in the context of browsable high-resolution content [131]. Modern games already make extensive use of high-resolution textures, and future games will continue this direction. For architectural purposes, virtual walkthroughs, or panorama shots, high-resolution imagery is often a necessity and renders the view experience more convincing. Avoiding expensive and time-consuming acquisition setups, therefore, is a crucial benefit.

Our work addresses the following challenges:

- Establishing reliable correspondences between photographs in unordered photo collections, even for cases where direct feature matching fails;
- Artifact free blending of (potentially overlapping) images at different resolution, taken with different cameras, differ-

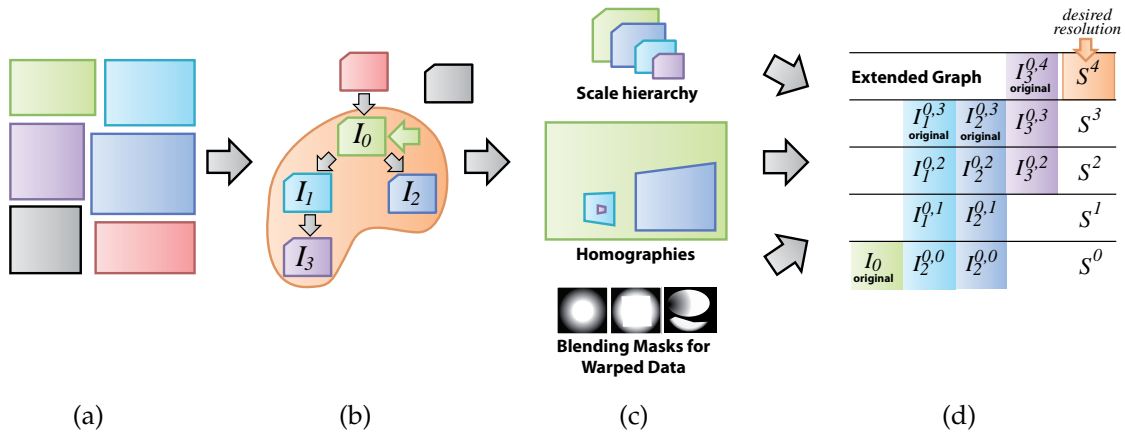


Figure 11.: Overview of our algorithm [74, 77]. (a) An unstructured set of images is transformed into (b) a dependency graph with scale relations. One image is chosen and (c) its children are adjusted: Homographies are established, blending masks computed, and an image hierarchy is created according to scale. (d) An optimization-based detail synthesis step then successively produces details at synthesis levels  $S^i$  by relying on information of scale  $i$  until the desired resolution is met.

ent focal length, white balancing, or color aberrations, or structural mismatches;

- Detail transfer and enhancement by information exchange between photos where no specific details are available.

Figure 11 shows an overview of our system that adds high-resolution details to low resolution content. Starting with a set of unordered images, Figure 11a, we extract prominent features and use these to establish parent-child relationships between images, Figure 11b. A child contains details of a parent image and we can derive scale factors indicating the resolution gain when relying on the children’s content, Section 4.2.

Next, a user selects an image to be augmented by details. In theory, we could simply project the children into the selected image, but this would lead to visible artifacts. Therefore, we adjust these images to make merging successful, Figure 11c. More precisely, we remove objects from the child images not visible in the parent image, adjust the colors and use a blend mask to hide the transition between the chosen and the detail images, Section 4.3.

Finally, parts not covered by the input images receive details using our constrained multiscale texture synthesis algorithm, Figure 11d. The synthesis involves a discrete optimization procedure to add new details at various output resolution levels, Section 4.4. We evaluate the algorithm on several test scenes in Section 4.5 and conclude with a brief discussion, Section 4.6.

## 4.2 DEPENDENCY GRAPH CONSTRUCTION

In order to create the high-resolution output, accurate information about the image relationships is needed, i.e., whether one image conveys details of another, if they are overlapping in the output image domain, or if they are not related at all. This is done fully automatically, the user only needs to chose the image he wants to augment with additional information.

Given a collection of photographs, we start by extracting feature points in each image. We use the SIFT keypoint detector [158] because of its invariance under affine image transformations. Especially scale-invariance proves useful for our task.

## 4.2.1 Parents Finding

We restrict each child to have one parent, but each parent can have several children. In a situation where an image  $\mathbf{I}_x$  is contained in an image  $\mathbf{I}_y$ , which in turn is contained in a third image  $\mathbf{I}_z$ , we want to avoid associating  $\mathbf{I}_x$  directly to  $\mathbf{I}_z$ . Instead, we do not want to skip relations and aim at establishing  $\mathbf{I}_x$  as a child of  $\mathbf{I}_y$  and  $\mathbf{I}_y$  as a child of  $\mathbf{I}_z$ .

Before selecting a parent for an image  $\mathbf{I}_c$  in the image collection, we first create a set of potential parents  $\{\mathbf{I}_p\}_c$  by comparing all photos with  $\mathbf{I}_c$ . We rely on the SIFT feature descriptors to find correspondences between image pairs  $\mathbf{I}_c$  and  $\mathbf{I}_p$  in order to establish reliable parent-child relationships between images. A match between features is considered valid if the euclidean distance in feature space between a feature vector in  $\mathbf{I}_c$  and its nearest neighbor in  $\mathbf{I}_p$  is smaller than 0.49 times the distance to the second nearest neighbor. An evaluation on the influence of this parameter can be found in [158]. If the number of all matched features between  $\mathbf{I}_c$  and  $\mathbf{I}_p$  is below a threshold  $\tau_m = 10$ , the images are considered unrelated. This threshold is rather uncritical, other works propose to use up to 20 [219], but this is already a very conservative number, in order to remove false positives, but can easily create false negatives.

If the two images are related, we try to establish a homography  $\mathbf{H}_{\mathbf{I}_c \rightarrow \mathbf{I}_p}$  between them warping  $\mathbf{I}_c$  into the image domain of  $\mathbf{I}_p$  using RANSAC [81] and DLT [110]. Other registration methods could be used, but this one worked particularly well and proved robust enough for our purpose. We denote  $\mathbf{I}_c^p := \mathbf{H}_{\mathbf{I}_c \rightarrow \mathbf{I}_p} \mathbf{I}_c$ .

Whenever it is possible to establish a homography from  $\mathbf{I}_c$  to  $\mathbf{I}_p$ , we add  $\mathbf{I}_p$  to the set of potential parents  $\{\mathbf{I}_p\}_c$ . To find the most appropriate parent, we project  $\mathbf{I}_c$  into each possible parent image  $\mathbf{I}_p \in \{\mathbf{I}_p\}_c$ , resulting in the warped image  $\mathbf{I}_c^p$ . The area of  $\mathbf{I}_c^p$  should be maximal in order to avoid skipping relations, as

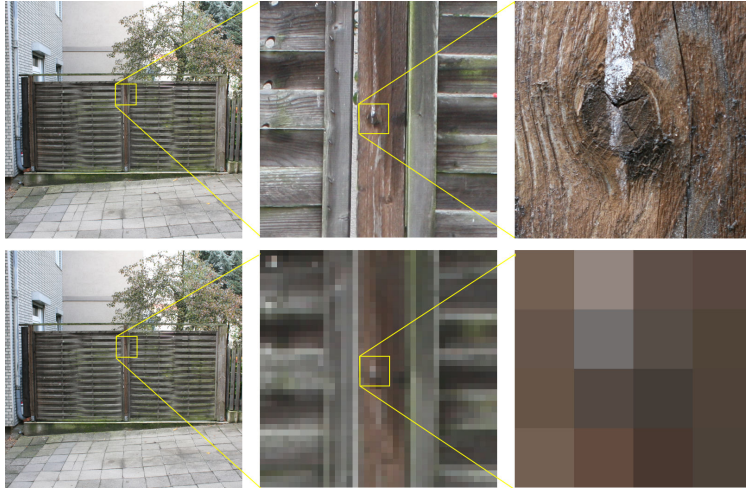


Figure 12.: The hierarchical nature of our dependency graph enables us to find the correct position for details derived from the input images at arbitrary scales. This would not be possible with direct feature matching, as the details could be arbitrarily small in the original image, even smaller than a single pixel. Top: Resulting zooms from our algorithm using six input images taken with different zoom factors. Bottom: Original image.

indicated earlier. More precisely, we compute the parent index for  $\mathbf{I}_c$  using the formula:

$$parentindex = \underset{p}{\operatorname{argmax}}(A(\mathbf{I}_c^p)), \quad (4.1)$$

where  $A$  is the area (number of pixels) of  $\mathbf{I}_c^p$  in  $\mathbf{I}_p$ . We impose that  $A(\mathbf{I}_c^p) < A(\mathbf{I}_p)$ , otherwise, the potential parent  $\mathbf{I}_p$  could also be a child of  $\mathbf{I}_c$ . If  $\mathbf{I}_c^p$  is not fully contained in  $\mathbf{I}_p$ , we mask out the pixels outside the valid region.

Having parent information available for each image we create a complete dependency graph for the whole image collection, Figure 11b. This rearrangement into the dependency graph structure allows us to establish correct homographies to every ancestor of each node. This would not be possible with direct feature matching, as current feature descriptors are only scale-invariant up to a certain amount of very few octaves in practice. An example for correct detail placement using our dependency graph is given in Figure 12.

#### 4.2.2 Preparing detail candidates

In the next step, the user is asked to choose the wanted root image  $\mathbf{I}_0$ , and we extract the corresponding subgraph from the dependency graph for further processing, Figure 11b. Due to possibly different resolutions of the input images, we need to



find out the amount of detail that can be added to a parent image  $\mathbf{I}_p$  by each of its children images  $\mathbf{I}_c$ . For this, we determine how much more resolution the warped image  $\mathbf{I}_c^p$  offers with respect to  $\mathbf{I}_p$ .

Let  $r_{c,p} = \frac{A(\mathbf{I}_c)}{A(\mathbf{I}_c^p)}$  be the ratio between the original amount of pixels in  $\mathbf{I}_c$  and the number of pixels occupied in  $\mathbf{I}_p$  by the warped image  $\mathbf{I}_c^p$ . If  $r_{c,p} \leq 1$ , the resolution of  $\mathbf{I}_c$  is considered insufficient to add information to  $\mathbf{I}_p$ . Basically, its pixels are larger than those in  $\mathbf{I}_p$ . In this case, we can remove  $\mathbf{I}_c$  from the dependency graph and make its children the new children of  $\mathbf{I}_p$ . Repeating this process recursively in a top-down manner removes all false details from the dependency graph.

The established dependency graph and warps will facilitate the later detail synthesis. Further, such dependency relations provide much algorithmic flexibility and can also be useful in other contexts. E.g., scaling and concatenating the homographies of each node and its ancestors and warping the images accordingly defines a higher resolution panorama image following standard techniques in [236]. By storing the scale ratio  $s = \lfloor r_{c,0} - r_{p,0} \rfloor$  between each child and its parent on the edges, the dependency graph becomes similar to an exemplar graph, presented in [107], but was fully automatically created.

### 4.3 DETAIL TRANSFER

Our algorithm will improve the quality of the selected root image using its descendents in the dependency graph. Because the child images may have been taken at different moments in time, with different cameras and from slightly different locations than the root image, we need to adjust and modify their content to better fit in the root image. We remove or adjust regions with a strong photometric inconsistency, adjust the color, and find a blending mask to create a good transition between the inserted element and the original image. The treatment described in this section is recursively applied to all children in the dependency graph in a top-down manner.

After these adjustments, it is possible to project all child images into an upscaled version of the root image and it leads to artifact-free transitions. This allows us to enhance the upscaled image with the captured details. In Section 4.4, we will present a detail synthesis algorithm to improve not only the covered areas, but the entire root image.

#### 4.3.1 Mismatch Removal

Detail images do not need to be taken from exactly the same viewpoint or at the same time. As a result artifacts such as parallax effects, temporal artifacts or objects might appear, or disappear, in the detail images. If no further knowledge is given of the content, our only option is to conservatively estimate similar regions in the parent and the child image and mask out dissimilar regions.

To remove the influence from small scale misalignments and noise, which should not be visible later on, we apply the homography matrix  $\mathbf{H}_{\mathbf{I}_p \rightarrow \mathbf{I}_c}$  to warp  $\mathbf{I}_p$  into the image domain of the child. Next, we blur both images, the parent and child image, with a Gaussian filter with a standard deviation of  $\sigma = 2$  pixels. We compute the Sum of Squared Differences (SSD) for each  $5 \times 5$  window around each pixel and only save the largest value of each color channel. Thresholding the resulting image reveals our final mask, marking the regions usable for further processing. Using an empirically estimated threshold of  $\tau_{SSD} = 0.35$  worked well in our test cases, the images are all scaled to lie in the range  $[0, 1]$ . We tried other methods as well, e.g., the mean-removed normalized cross correlation, which has been proposed by Goesele *et al.* [94] for a similar purpose. But the results were not always as satisfactory even with an optimized threshold, see Figure 13 for a comparison.

#### 4.3.2 Structural Adaptation

There are many cases where one actually knows, that no new objects appear or disappear in the detail images, e.g. if a static scene was photographed. In these cases it is not necessary to mask out parts of the child images. Instead we carefully adapt their content to remove structural misalignments. In order to not introduce additional artifacts we aim at matching only the most salient structures in the image and interpolate the rest. These structures or edges are usually the main disturbances when compositing two images [125].

Consider the basic task of stitching together two images  $\mathbf{I}_S$  and  $\mathbf{I}_T$ , e.g. the child and parent image, which have already been roughly aligned as described in Section 4.3.1 and overlap in an area called  $\Omega$ , Figure 14. The partitioning seam inside  $\Omega$  is called  $\partial\Omega$  with  $\partial\Omega_S$  and  $\partial\Omega_T$  depicting the pixels along the border in  $\mathbf{I}_S$  and  $\mathbf{I}_T$  respectively. Our structural adaptation algorithm [77] proceeds in six steps:

1. An optimal partitioning is computed between the roughly aligned images  $\mathbf{I}_S$  and  $\mathbf{I}_T$ .

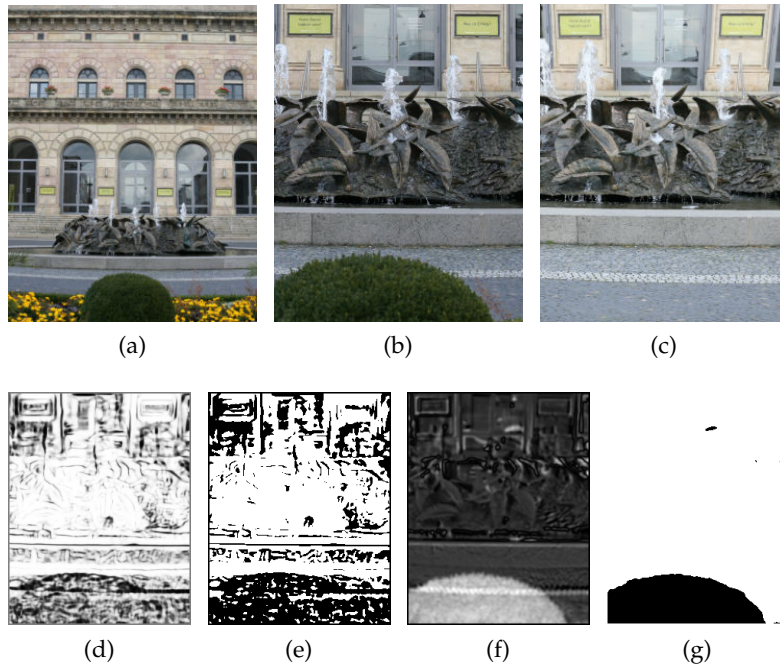


Figure 13.: Object Removal. (a) Parent image, (b) cropped region of interest from parent image, (c) warped child image cropped to region of interest. Note the strong parallax - the pavement visible in the detail image is occluded by a bush in the parent image. (d) The normalized cross correlation for  $5 \times 5$  regions around each pixel between the parent and child image, as proposed in [94], does not provide a good clue which parts of the image belong to the same object. (e) The resulting mask after thresholding is unsatisfactory. (f) The sum of squared differences gives a better indication of differing regions. (g) The resulting mask after thresholding excludes the unwanted object pretty well.

2. Features along the partitioning seam  $\partial\Omega$  are matched and brought into alignment.
3. The outgoing edges along these features are traced and brought into alignment.
4. The sparse deformation field derived from the matching is propagated throughout the area of the source image  $I_S$  which is warped accordingly.
5. To take the deformation into account, we compute a new optimal partitioning.
6. The color values of  $I_S$  are adjusted subject to a constraint Poisson equation.

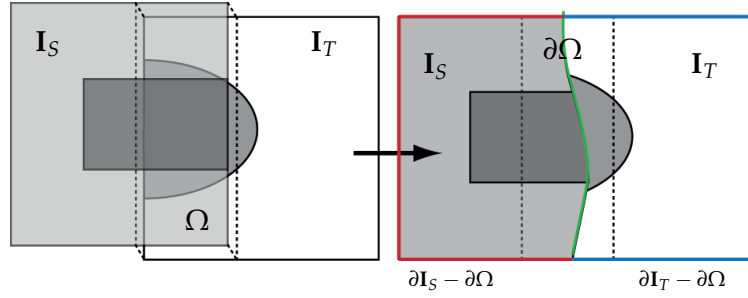


Figure 14.: Optimal partitioning for two overlapping images: (left) The images  $I_S$  and  $I_T$  have been globally aligned and overlap in the area  $\Omega$ . (right) The partitioning divides  $\Omega$  into two parts. Based on the found seam  $\partial\Omega$  the images are combined into a common image space. The border around the image excluding the seam  $\partial\Omega$  is denoted  $\partial I_S - \partial\Omega$  and  $\partial I_T - \partial\Omega$  respectively.

#### 4.3.2.1 Optimal Partitioning

We employ the *Drag-and-Drop Pasting* method [126] to find an optimal partitioning seam in  $\Omega$ . Starting with an arbitrary path this iterative technique can find the seam optimizing the following energy function:

$$E(\partial\Omega, k) = \sum_{(x,y) \in \partial\Omega} ((I_T(x,y) - I_S(x,y)) - k)^2, \quad (4.2)$$

where  $k$  is the average color difference on the boundary seam  $\partial\Omega$  computed as the  $L_2$ -norm on the  $rgb$ -triplets.

For the case where  $I_S$  is fully surrounded by  $I_T$ , which is usually the case for object insertion tasks, we define a foreground object  $\Omega_{Obj}$  in  $I_S$  by applying the GrabCut technique of Rother *et al.* [201]. This defines a foreground area in  $I_S$  which may not be crossed by the optimal seam, Figure 15. If  $I_S$  only partially overlaps  $I_T$  we can force the seam to cross the boundary pixels of  $\partial I_S - \partial\Omega$  by enforcing  $I_S$  to belong to the foreground object except for the border pixels  $\partial I_S - \partial\Omega$  and  $\Omega$ , this way both cases can be treated in the same way. This seam provides a reasonable starting point for the feature tracing and matching applied in the following.

#### 4.3.2.2 1-D feature detection and matching

In our observation the most prominent artifacts are produced by mismatching salient edges in both images. Therefore, the first step is to detect these edges. We start by removing noise in the image by applying a bilateral filter [241]. Using the Canny edge detector [39] we find all important edges in the images and thin them out, to assure there are only two neighboring pixels for each edge pixel, except at crossings, which is beneficial for the latter edge matching. The same observation was made by Jia *et*

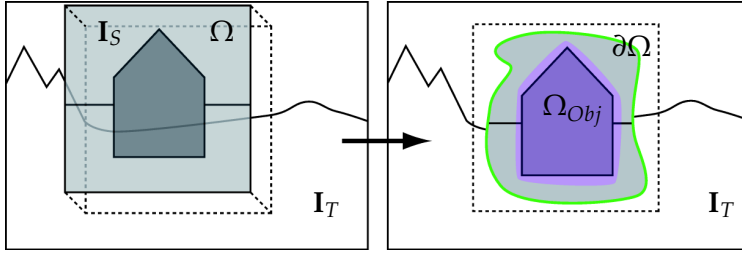


Figure 15.: Optimal partitioning for two images, one enclosing the other: (left) The source image  $I_S$  is completely surrounded by its target  $I_T$ . (right) A foreground region  $\Omega_{Obj}$  (lilac) is defined through which the seam may not pass, and the optimal seam is computed around it (green).

al. [125] and we follow their idea of aligning the salient edges along the seam. Assuming, without loss of generality, that there are  $n$  edges found along  $\partial\Omega_S$ ,  $m$  edges along  $\partial\Omega_T$  and  $n \geq m$ , an optimal edge matching can be found by dynamic programming:

$$E' = \min \sum_{0 \leq i < m} (p_T(i) - p_S(k_i))^2, \quad (4.3)$$

$$\text{s.t. } 0 \leq k_0 < k_1 < \dots < k_{m-1} < n,$$

where  $p_S(\cdot)$  and  $p_T(\cdot)$  are the pixel positions of the salient edges along  $\partial\Omega_S$  and  $\partial\Omega_T$ , respectively. For each of the matched edge pixels, a deformation vector  $\mathbf{d}$  pointing from its pixel position along  $\partial\Omega_T$  towards the position of its match along  $\partial\Omega_S$  is defined as a constraint for a deformation field  $\mathbf{D}$  for  $I_S$ , Figure 16.

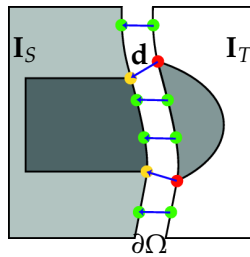


Figure 16.: Structure misalignments might still persist along the border of an optimal partitioning. To remove these misalignments we compute the most salient edge pixels (red and yellow) along  $\partial\Omega$  and compute a deformation vector for each of these. Additional zero vectors (green) are added to prevent excessive deformations.

#### 4.3.2.3 1.5-D feature matching

Once a matching of the salient edge pixels along  $\partial\Omega$  has been computed, we need to propagate these deformations in a meaningful manner to the rest of the pixels. To restrict the deformation of  $I_S$ , we set additional zero deformation vectors  $\mathbf{O} = (0, 0)$  for those pixels along  $\partial\Omega$  that are 10% of the seams overall length

away from any previously computed deformation vector  $\mathbf{d}$ . The 10% are chosen empirically but give good results in our test cases. For the rest of the unassigned pixels along  $\partial\Omega$  we linearly interpolate the values of the two neighbouring deformation vectors to the left and to the right.

Matching of the salient edges avoids structural mismatches along the seam, one can think of this as  $C^0$ -continuity, but the edge direction can still change rather abruptly, so there is no real  $C^1$ -continuity along the edges. We will therefore trace the edges further into  $I_S$  and  $I_T$  and match these as well.

To trace an edge starting at the edge's pixel position  $p$ , we create an edge path  $\mathbf{P}$  of a preset length  $l$ , but even small values work already well in most cases. Due to the edge-thinning we can usually walk directly along the edges already found in Section 4.3.2.2 by the Canny edge detector [39]. In case of ambiguities we follow the strongest gradient strength. As a convention we will use  $\mathbf{P}_{I_A \rightarrow I_B}$  to denote the set of edge pixels in  $I_A$  starting at  $p$  and going in the direction of  $I_B$ , Figure 17.

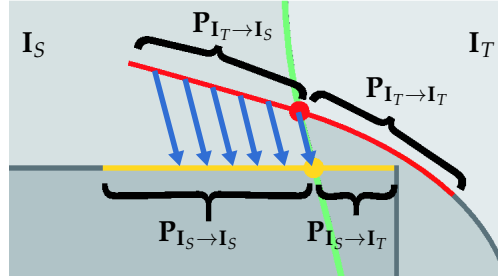


Figure 17.: Naming convention of the traced edges. This is basically a close-up of Figure 14. The in- and outgoing edges in  $I_T$  are marked in red, the respective edges in  $I_S$  are marked in yellow. The deformation on the seam  $\partial\Omega$  is propagated along  $\mathbf{P}_{I_T \rightarrow I_S}$  (blue arrows).

For each pixel position along  $\mathbf{P}_{I_T \rightarrow I_S}$  we set  $\mathbf{d}_{p_n} = \mathbf{P}_{I_S \rightarrow I_S}(n) - \mathbf{P}_{I_T \rightarrow I_S}(n)$ , if  $\mathbf{P}_{I_T \rightarrow I_S}(n)$  is available, where  $\mathbf{d}_{p_n}$  is the deformation vector at pixel position  $\mathbf{P}_{I_T \rightarrow I_S}(n)$ . The  $n$ -th pixel position in  $\mathbf{P}_{I_T \rightarrow I_S}(n)$  is matched with the  $n$ -th pixel position in  $\mathbf{P}_{I_S \rightarrow I_S}(n)$ .

#### 4.3.2.4 Deformation propagation

The rest of the deformation field  $\mathbf{D}$  to deform the source image  $I_S$  is filled as smooth as possible. Let  $\{p\}_{edge}$  be the set of edge pixels with an already defined deformation vector. We solve the following diffusion equation with Dirichlet boundary conditions:

$$\begin{aligned} \mathbf{D}^*(x, y) &= \mathbf{D}(x, y), \text{ if } (x, y) \in \{p\}_{edge} \\ \nabla^2 \mathbf{D}^*(x, y) &= 0, \text{ otherwise} \end{aligned} \quad (4.4)$$

where  $\nabla^2$  is the Laplace operator.

After minimization, each pixel in our image domain is associated with a deformation vector in  $\mathbf{D}^*$ . Performing an inverse mapping with bilinear interpolation on  $\mathbf{I}_S$ , we obtain the warped and structurally aligned image. As the image was deformed during this process, we compute a new optimal partition to assure that we are still given the optimal seam as described in Section 4.3.2.1. Everything outside this seam is masked out for further processing.

### 4.3.3 Color Adjustment

After taking care of the structural misalignments, varying white balance and exposure settings can cause color aberrations between parent and child images. In order to fix these, we use a recursive gradient domain fusion on the elements of the dependency graph.

Starting at the root we process the dependency graph in a top-down manner and for each child  $\mathbf{I}_c$  and its parent image  $\mathbf{I}_p$  we apply the inverse homography matrix  $\mathbf{H}_{\mathbf{I}_c \rightarrow \mathbf{I}_p}^{-1}$  again to warp  $\mathbf{I}_p$  into the image domain of the child. This allows us to add a one-pixel border around the previously computed mask of the child image by sampling colors from  $\mathbf{H}_{\mathbf{I}_c \rightarrow \mathbf{I}_p}^{-1} \mathbf{I}_p$ . We then find the image that best matches the gradients of the child image while respecting the sampled boundary color values. This can be expressed as a Poisson equation with Dirichlet boundary conditions following [184], see Section 2.12. The boundary conditions are given by the one-pixel border derived from  $\mathbf{H}_{\mathbf{I}_c \rightarrow \mathbf{I}_p}^{-1} \mathbf{I}_p$ , while the guidance field  $\mathbf{v}$  for the Poisson equation is given by the gradient of  $\mathbf{I}_c$ . A comparison with and without Poisson blending is given in Figure 18.

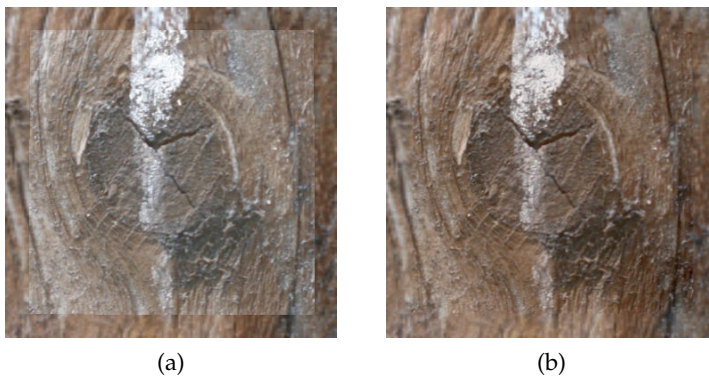


Figure 18.: Color Adjustment. (a) Without poisson blending, the detail child patch might differ in color from the low resolution parent image. (b) After poisson blending the colors are adjusted.

4.3.4 *Blending Mask*

Copying a child image directly into an upscaled version of the root image  $\mathbf{I}_0$  is likely to produce seams, even after poisson blending, due to the higher frequency bands present in the detail image. To make the insertion successful, we refine the computed mask by attributing opacity values to all child-image pixels that indicate how to blend the content with the output image. An easy way to extract a blending matte is to compute a distance map, also called grassfire transform [236]. For every pixel, the distance to the image center is computed. The larger the distance, the more transparent the pixel becomes. The downside of such solutions is that image content is not taken into account and the transition will be easily noticeable. Instead, we found that much better transitions are possible when exploiting the image content.

We first establish a *gradient map*  $\mathbf{G}_{c,a}$ ,  $a \in \{r, g, b\}$  for each color channel:

$$\mathbf{G}_{c,a} = \|\nabla \mathbf{I}_{c,a}\|_1 = |\nabla_x(\mathbf{I}_{c,a})| + |\nabla_y(\mathbf{I}_{c,a})|, \quad (4.5)$$

where  $\nabla_x(\mathbf{I}_{c,a})$  and  $\nabla_y(\mathbf{I}_{c,a})$  are the gradients in  $x$  and  $y$  direction of  $\mathbf{I}_c$  in color channel  $a$  respectively. Using the  $L^1$  norm in Equation (4.5) leads to faster changes of the blending values along edges, due to the triangle inequality, resulting in less distracting transitions than with the common  $L^2$  norm. We then establish a *gradient density map*  $\mathbf{G}'_{c,a}$  computing for every pixel  $(x, y)$  the least cost path to a border pixel of its mask, using dynamic programming [24], according to

$$\mathbf{G}'_{c,a}(x, y) = \min_{path} \left\{ \sum_{(u,v) \in path} \mathbf{G}_{c,a}(u, v) \right\} \quad (4.6)$$

We combine the gradient density map of all three color channels by saving only the maximum costs in  $\mathbf{G}'_c$ . Using the separate maps  $\mathbf{G}'_{c,a}$  for each color channel would lead to unwanted color aberrations. The cost for pixels outside the mask are zero. Consequently, regions with only few color gradient changes will be assigned a relatively slow growing value from the border of the mask to the pixel of interest. In regions with strong edges the cost value will rise faster, as slow blending could produce visible ghosting artifacts or disturbing blur in these areas, Figure 19. In addition, pixels closer to the patch center will usually receive higher weights than those close to the border. The final blending mask is then computed using a combined thresholding and scaling:

$$\alpha(x, y) = \min\left(1.0, \frac{\mathbf{G}'_c(x, y)}{\tau}\right) \quad (4.7)$$



where  $\tau$  is kept to 0.4 of the maximum value of  $\mathbf{G}'_c$  throughout our examples, which turned out to be a good tradeoff between the speed of the transition and preserved area of the image. For high frequency textures, it is beneficial to multiply  $\alpha$  with a Gaussian falloff function to slow down the transition in these areas.

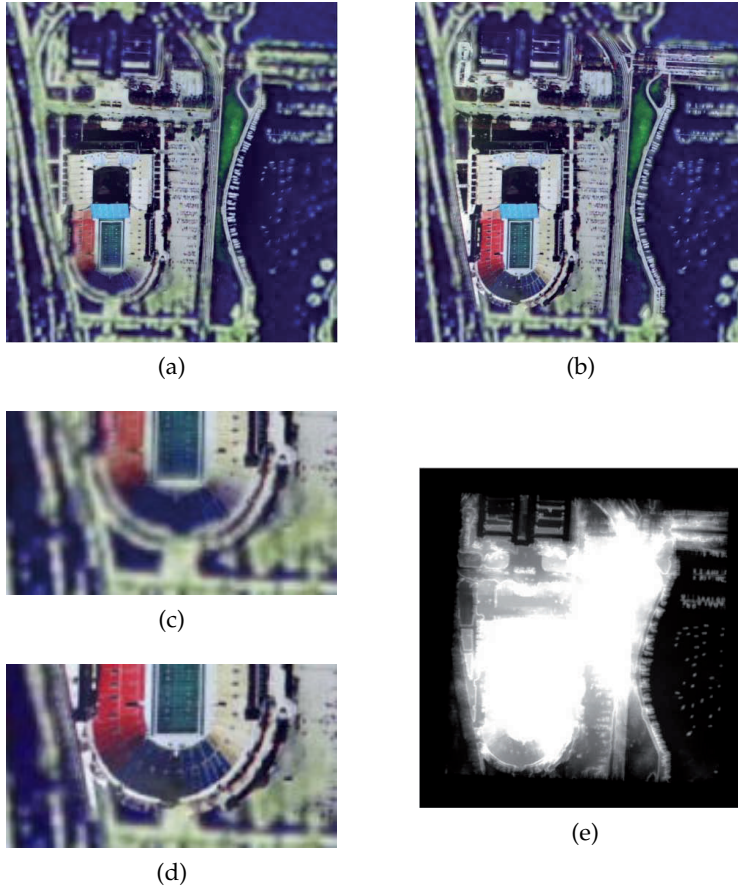


Figure 19.: Blending example: Combining a high- and low-resolution image. (a) Blending using a Gaussian falloff mask. (b) Our edge-aware result. (c) and (d) Close-up of (a) and (b), respectively. (e) Visualization of our final blending mask  $\alpha$ .

#### 4.4 CONSTRAINED MULTISCALE DETAIL SYNTHESIS

To add plausible details to the root image  $\mathbf{I}_0$  we will make further use of the derived scale relationship and image adaptation. In the following, we will describe how to use the scale relationships to derive a multiscale dependency graph. This is crucial for our detail synthesis algorithm, described in depth in Section 4.4.2. The detail synthesis works hierarchically by establishing matches between images of corresponding resolution levels. Starting with the original resolution of  $\mathbf{I}_0$ , we successively up-scale this image. After each upsampling, a blending and a detail

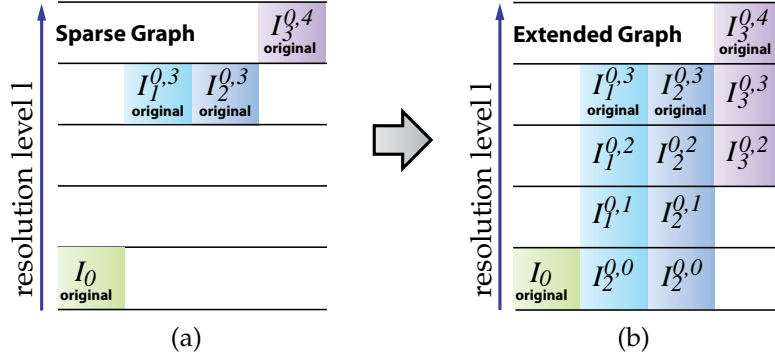


Figure 20.: Extended Dependency Graph: (a) Once the scale relations are known, the warped images can be arranged in a sparse dependency graph. (b) By creating a Gaussian image pyramid out of each image, more exemplars can be added to each level. All exemplars of the same level are used for the later texture synthesis steps.

synthesis step is applied, where data is also used from other images of the corresponding level.

#### 4.4.1 Extended Dependency-Graph

It is easier for the synthesis to work with power-of-two scale factors. Hence, we determine a *resolution level*  $L$ , representing a scale factor of  $2^L$ . For each detail image  $\mathbf{I}_i$ ,  $L$  is maximized such that the original resolution ratio  $r_{i,0} = \frac{A(\mathbf{I}_i)}{A(\mathbf{I}_i^0)}$  between  $\mathbf{I}_0$  and  $\mathbf{I}_i^0$  is still larger than  $2^L$ :

$$\operatorname{argmax}_L (2^L \geq r_{i,0}), L \in \mathbb{N}_0 \quad (4.8)$$

Because the original input images represent only a sparse refinement candidate set, we create a Gaussian image pyramid out of each  $\mathbf{S}\mathbf{H}_{\mathbf{I}_i \rightarrow \mathbf{I}_0} \mathbf{I}_i$  where  $\mathbf{S}$  is a scaling matrix scaling by a factor of  $2^L$  to preserve the details in  $\mathbf{I}_i$  before creating the image pyramid. One downsampling operation, reducing width and height by a factor of two, transforms an image of level  $l$  into an image of level  $l - 1$ , Figure 20. Having defined these multi-resolution representations, we will denote the warped and accordingly scaled image  $\mathbf{I}_i$  at level  $l$  as  $\mathbf{I}_i^{0,l}$ . Similarly, the blending masks are also denoted  $a_i^l$  to reflect the corresponding scale. Adding  $\mathbf{I}_0$  to each synthesis level helps to refine regions with different colors than those represented in the detail images, e.g., in a panorama one seldomly takes detail shots of the blue sky. Below we describe how to make use of the previously derived representations in our multiscale texture optimization framework.

#### 4.4.2 Multiscale Texture Synthesis

Our algorithm builds an output image pyramid  $\mathbf{S}^0, \mathbf{S}^1, \dots, \mathbf{S}^T$  in a coarse-to-fine order, where  $\mathbf{S}^T$  is the final image of the desired output resolution. The images  $\mathbf{S}^t$  are not represented by color values, at a pixel position  $(x, y)$ , but rather store coordinate information in the form  $\mathbf{S}^t(x, y) = (u, v, i, l)$ , where  $(u, v)$  are pixel coordinates,  $i$  is the image id, and  $l$  is the scale level. We will use the notation  $*\mathbf{S}^t$  to refer to the actual color image of  $\mathbf{S}^t$ , which is saved separately and updated on demand. Unlike traditional texture synthesis methods, we do not start with a  $1 \times 1$  image or random noise patterns, but rather start by refining the root image  $\mathbf{I}_0$ .

Each level  $\mathbf{S}^t$  is generated by (1) upsampling the image  $\mathbf{S}^{t-1}$ , (2) optionally blending the detail images  $\mathbf{I}_i^{0,t}$  with the resulting color image  $*\mathbf{S}^t$  and (3) locally refining the image by a detail synthesis algorithm.

##### 4.4.2.1 Upsampling

Instead of upsampling color from the last synthesis step, we up-sample coordinates. Specifically, we adopt the idea of Lefebvre *et al.* [139] and ascend in the hierarchy to a higher-resolution level, if available. Hence, we introduce new details even before refining the upsampled image. For  $\mathbf{S}^{t-1}(x, y) = (u, v, i, l)$ , the upsampled patch is defined by:

$$\mathbf{S}^t(2x + \lambda_x, 2y + \lambda_y) := (2u + \lambda_x, 2v + \lambda_y, i, l + 1) , \quad (4.9)$$

$$\text{with } \begin{pmatrix} \lambda_x \\ \lambda_y \end{pmatrix} \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

If a higher-resolution level is not available, we simply copy the content of  $\mathbf{S}^{t-1}(x, y)$  to all four corresponding pixels of the next resolution level.

##### 4.4.2.2 Blending

As described in Section 4.3, we have all the information available to directly blend entire child images into the synthesized image at each level. This is especially helpful in the context of multi-scale panorama images. In this case, we compute a new solution  $*\mathbf{S}^t$  from the upsampled coordinates of  $\mathbf{S}^{t-1}$ . To add the specific details from our input images, we blend each child  $\mathbf{I}_i^{0,t}$  with  $*\mathbf{S}^t$  using:

$$*\mathbf{S}^t = \alpha_i^t \mathbf{I}_i^{0,t} + (1 - \alpha_i^t) (*\mathbf{S}^t), \quad (4.10)$$

where  $\alpha_i^t$  is the previously computed blending mask. The blending order of the children depends on the resolution level  $L$  com-

puted in Section 4.4.1. The higher the relative resolution, the later it is added.

#### 4.4.2.3 Detail Synthesis

In the last section we augmented our upsampled image at specific, known position with details from the input images. For the other regions, we will try to find plausible details by texture synthesis.

For each synthesized pixel  $(x, y)$  in  $\mathbf{S}^t$ , the detail synthesis step seeks to find a pixel position  $m(x, y)$  in any detail image  $\mathbf{I}_i^{0,t}$  of level  $t$  whose local  $5 \times 5$  neighborhood  $\mathbf{N}(m(x, y))$  best matches the  $5 \times 5$  neighborhood  $\mathbf{N}(x, y)$  in  $\mathbf{S}^t$  centered at  $(x, y)$ . A neighborhood  $\mathbf{N}(x, y)$  around a pixel position  $(x, y)$  consists therefore of 25 *rgb*-values. Using a larger neighborhood usually only increases computation time in hierarchical texture synthesis, as was already pointed out by several other authors [108, 139].

Basically, our goal of synthesizing new details can then be seen as the minimization of an error functional, which is determined by mismatches of input/output neighborhoods:

$$E := \sum_{(x,y) \in \Omega^t} \|\mathbf{N}(x,y) - \mathbf{N}(m(x,y))\|_2, \quad (4.11)$$

where  $\Omega^t$  is the image domain of  $\mathbf{S}^t$ .  $E$  measures the sum of all neighborhood differences across the current image. Basically, if neighboring pixels had neighboring matches, this functional would be minimal. In practice, even if pixels are neighbors in the output image, the best matches might be very different and we need to apply an optimization procedure to improve the original image.

In order to minimize the error functional in Equation (4.11), we minimize the error for each level using a discrete two-step EM-like (Expectation/Maximization) solver, similar in spirit to [108, 135]. A visual illustration of the process is also given in Figure 21. In the M-step, the set of output pixels at  $(x, y)$  remains fixed and a set of  $n$  best matching input neighborhoods  $\{\mathbf{N}(m(x, y)^k)\}$  is found per pixel position  $(x, y)$ ,  $k \in [1, \dots, n]$  denotes the index of the  $k^{\text{th}}$  best matching neighborhood. In practice, we use  $n = 3$ , Han *et al.* [107] use  $n = 2$  which we found to be a too small number for sufficient results, Tong *et al.* [242] originally proposed to use  $n \leq 11$ , but we did not experience any visual improvement beyond  $n = 3$ .

In the E-step, the set of best matching input neighborhoods  $\{\mathbf{N}(m(x, y)^k)\}$  remains fixed while we optimize for  $*\mathbf{S}^t(x, y)$  and, hence, modify  $\mathbf{S}^t$ . We look at all pixels at position  $(x, y) + \Delta$  in a  $3 \times 3$  neighborhood (candidate neighborhood) around  $(x, y)$ . We then gather all their best matching neighborhood centers

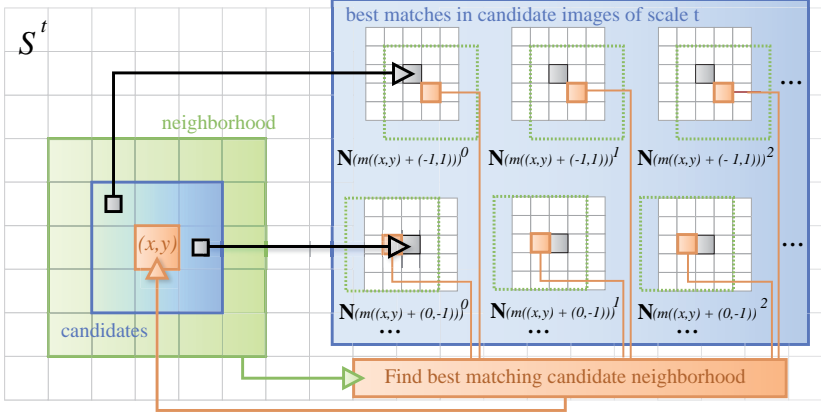


Figure 21.: Optimization procedure: Color values are optimized by improving coherence of neighboring pixels. For each pixel from a  $3 \times 3$  neighborhood around  $(x, y)$ , its  $5 \times 5$  neighborhood is extracted and the  $k$  best matches  $\mathbf{N}(m((x, y) + \Delta)^k)$  are found in the candidate images (gray grids on the right). The neighborhoods from the shifted center pixel  $\mathbf{N}(m((x, y) + \Delta)^k - \Delta)$  (dotted region around red pixels on the right) are then compared to  $(x, y)$ 's original neighborhood  $\mathbf{N}(x, y)$ , and the pixel at position  $(x, y)$  is replaced with its best match.

$\{m((x, y) + \Delta)^k\}$ . To chose the new value for  $\mathbf{S}^t(x, y)$ , we compare all candidate neighborhoods  $\mathbf{N}(m((x, y) + \Delta)^k - \Delta)$  to the neighborhood  $\mathbf{N}(x, y)$  around  $(x, y)$ , as illustrated in Figure 21. Let  $\mathbf{N}(m((x, y) + \Delta_{min})^j - \Delta_{min})$  be the neighborhood that minimizes the difference to  $\mathbf{N}(x, y)$ . We then associate with  $\mathbf{S}^t(x, y)$  the value of  $m((x, y) + \Delta_{min})^j - \Delta_{min}$ , i.e., position, index and level.

The E- and M-step are repeated until a minimum is reached, i.e., the best matching neighborhoods  $\mathbf{N}(m(x, y))$  do not change anymore from one iteration to the next, or a maximum number of iterations is reached. We use up to four iterations in our implementation.

To summarize this step, the detail synthesis tries to adjust the image in such a way that every local neighborhood around each pixel resembles a neighborhood in one of the input images of the current level. This optimization does not affect the blended areas, as for these perfect matches can be found and they will therefore not be replaced with other values, except at the boundaries for a better merging with the rest of the image.

#### 4.4.2.4 Accelerating Neighborhood Matching

For faster computations, we use an approximate nearest-neighbor search [16] to find the  $k$  best-matching neighborhoods  $\mathbf{N}(m(x, y))$  in all  $\mathbf{I}_i^{0,t}$  for each  $\mathbf{N}(x, y)$ . We did not adopt k-coherence [242] in this step, as this might restrict us to too few good matches

and also complicates the integration of the blending step, as the blending affects only  $*S^t$  and  $S^t$  stays unaffected until the detail synthesis step.

We further project all neighborhoods into a truncated principal component analysis (PCA) space [127]. The PCA basis for each level  $t$  is constructed by using all neighborhoods from all admissible candidates  $I_i^{0,t}$ . We automatically derive the number of needed eigenvectors by truncating as soon as the eigenvalues drop to 0.5% of the largest eigenvalue. This usually results in 9 to 15 eigenvectors used for projection and the results are visually indistinguishable from using all 75 eigenvectors.

#### USER SELECTION OF ROOT IMAGE

Photographers usually take a lot of similar pictures of a scene and choose the best one afterwards. We can augment this selected photo with a slight change to our algorithm. We first establish the dependency graph as usual. Then all root images except for the one selected are deleted and a homography of their child nodes to the selected image is computed. Upon success, the whole subtree is added as a child to the selected image, as its established relations remain valid. Otherwise the whole subtree is removed from further consideration.

#### 4.5 RESULTS

We have evaluated our system with several test collections. The synthesis itself takes about 30 seconds for an image of size  $256 \times 256$  on an AMD Athlon 64 X2 Dual Core Processor 4800+, with only one core used, and 3GB of RAM. It scales linearly with the number of output pixels and approximately logarithmically with the number of input pixels, i.e., the exemplars. Four iterations have been applied during the optimization step of each level in all examples.

#### RELATIONSHIP RECONSTRUCTION

To test the relationship reconstruction presented in Section 4.2, we created a database of 46 images which we took from 7 different scenes and also used different camera models, a subset is shown in Figure 22. This way we could establish a ground



Figure 22.: Some images from our ground truth data set to test our relationship reconstruction approach, consisting of 46 images from 7 different scenes.

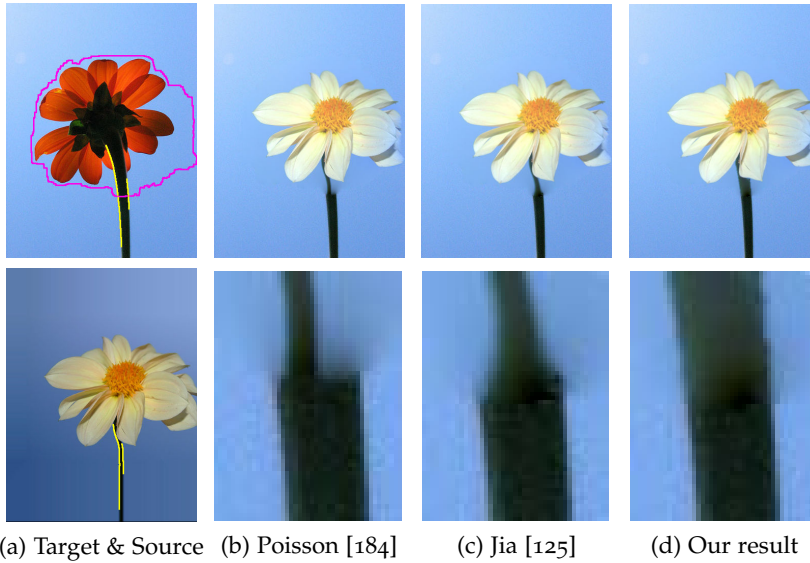


Figure 23.: Structural adaptation test: The blossom of the white flower replaces the red one. (a) The traced edges of the stem are marked in yellow, the optimized seam in magenta. (b) After roughly aligning the images, Poisson blending [184] reveals a strong structural misalignment at the flower’s stem. (c) The technique of Jia *et al.* [125] is able to match the corresponding edges, but results in a rather disturbing structural transition at the seam. (d) Our edge tracing method automatically propagates the necessary deformation of the stem more faithfully into the rest of the image. The according deformation of the blossom is unnoticeable to the human observer.

truth dependency graph to which we compared the result of our algorithm. Our system created correct relationships for 91.3% of the images, four images were excluded by the system, but none were falsely assigned. The whole process took about 725 seconds, as every image had to be matched to each other image in our current implementation.

#### STRUCTURAL ADAPTATION

We first tested our structural adaptation algorithm by replacing the blossom of a flower with another blossom and use our method to adjust the stem in order to create a plausible transition between the two images, Figure 23. In a first step the images are roughly aligned by hand. The yellow pixels in Figure 23a show the traced edges, the seam is shown in magenta in the top left image and is used for partitioning in this experiment. We compare our method to two other established techniques for seamless image stitching, namely *Poisson Blending* by Pérez *et al.* [184], Figure 23b, and *Image Stitching Using Structure Deformation* by Jia *et al.* [125], Figure 23c. In the bottom row, we show

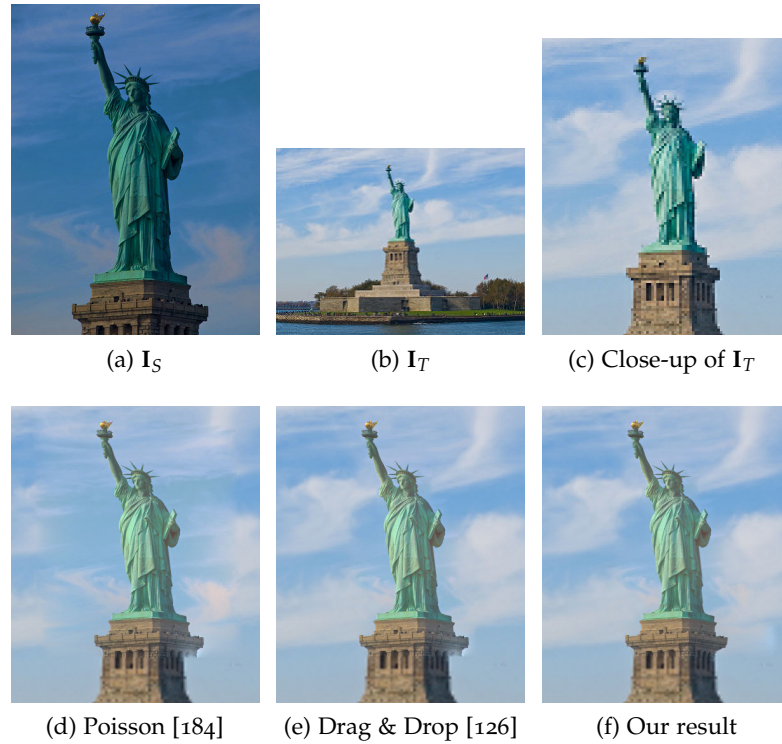


Figure 24.: Structural adaptation test with different resolution levels: (a) The source image  $I_S$  is merged into (b) an image  $I_T$  with lower resolution level. (c) Close-up of the target image  $I_T$ . (d) Close-up of the result using Poisson blending by Pérez *et al.* [184]. Note the mismatches at the pedestal and in the clouds. (e) Close-up of the result with an optimized seam using Drag & Drop Pasting [126]. The clouds look better, but the mismatch at the pedestal is still present. (f) Close-up of our result which can also handle the structural inconsistencies.

close-ups of the transition area along the stem. The images in Figure 23b show the result using only the Poisson Blending technique of Pérez *et al.* Although the color discrepancy is alleviated, the transition between the two stems is clearly visible due to their differing width. Strong color bleeding artifacts are the result. The method of Jia *et al.*, Figure 23c, nicely adjusts the stem along the seam, but the transition area is still annoyingly visible due to the fact that the sparse deformation constraints are only computed along the seam and interpolated into the rest of the image. Using the same deformation vectors along the seam, but our edge tracing method to match the interior edges, we can create a much more natural looking transition without noticeably deforming the blossom, Figure 23d.

To test how well this approach works with different resolution levels we used two images of the statue of liberty, one wide-angle shot and one with a close-up view of the statue, Figure 24. Poisson blending [184] can adjust the color of the source image,



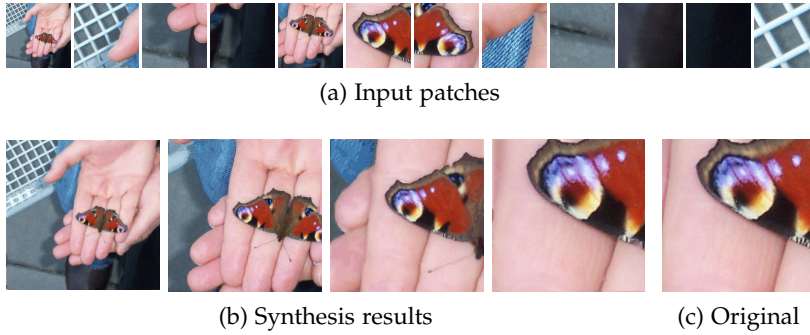


Figure 25.: Ground truth test for our detail synthesis: (a) The high resolution image,  $2048 \times 2048$  pixels, on the left is downsampled to  $128 \times 128$  pixels, and distinctive parts have been cut out at various zoom levels, which are used as input to the detail synthesis algorithm. (b) The overall appearance is well preserved by our detail synthesis algorithm when upsampling the image again. (c) For comparison, the same part taken from the original high resolution image.

but structures like the clouds or the basement cannot be handled properly, Figure 24d. Using *Drag & Drop Pasting* [126] an optimal seam is created so that the transition in the clouds is less visible, Figure 24e. The structural misalignments, however, are still not handled well. Our approach, Figure 24f handles both color and structural discrepancies sufficiently well: visible seams are removed while the applied deformation is hardly noticeable.

#### GROUND TRUTH SYNTHESIS

To test our detail synthesis algorithm without the blending step we created another ground truth test case. Starting from a high resolution image of size  $2048 \times 2048$  pixels, we cut out significant parts of the image at various resolution levels and downsampled the image to  $128 \times 128$  pixels. We then upsampled the image again using these small patches as input to our detail synthesis algorithm. All patches were of the size  $128 \times 128$  or  $256 \times 256$ . Figure 25a shows the input patches used and Figure 25b shows several zoom shots created by our detail synthesis algorithm. As a comparison Figure 25c shows the same region from the high resolution image. The overall appearance is well preserved by our detail synthesis algorithm even though each pixel has been enlarged to 256 pixels. Only a slight change in the position of some of the details is visible, e.g. the small blue dots on the butterfly wings. In our complete algorithm the blending step would take care of the correct placement for given details.

#### MULTISCALE PANORAMA

Figure 26 shows a large-scale panorama created from 9 input images at different scales. In contrast to previous panorama-



(a)



(b)



ours

original

ours

original

(c)



ours

original

ours

original

(d)

Figure 26.: Multiscale Panorama: Using 9 input images at various zoom levels, partly overlapping and of varying sizes between  $483 \times 525$  pixels and  $1086 \times 585$  pixels, our algorithm automatically establishes the dependency graph, scale relations, and blending masks to create a high-resolution panorama image. We upsampled the original panorama with a  $683 \times 512$  pixels resolution to  $5464 \times 4096$  pixels. (a) Thumbnails of all input images used. (b) Automatically generated panorama by our algorithm. (c) Two examples of regions incorporating information from the detail images and the respective part in the original panorama image. Notice that the given details have been faithfully included in the high-resolution panorama. (d) Two examples for enhanced details next to their respective parts from the original panorama image. Both examples show regions where no direct correspondence relation with respect to the input images existed. Our solution adds subresolution detail, invisible in the original input image (here, with 64 times more pixels). In many cases, plausible details are added by our algorithm, e.g., the solar panels on the roof (left). The texture synthesis step is especially useful for small scale and repetitive structures, such as leaves of trees. However, if no sufficient detail information is available from other parts of the input images, it is not always possible to reconstruct semantically meaningful structures, like the houses on the right.

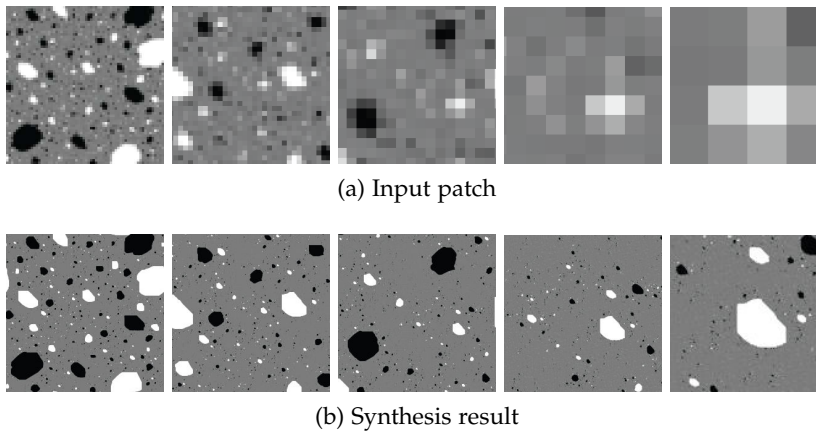


Figure 27.: Multiscale Texture Synthesis: A  $64 \times 64$  pixels input exemplar (upper left) is upsampled to  $2048 \times 2048$  pixels using our algorithm and exploiting self-similarity in the input image. (a) Various zoom levels of the original input patch. (b) The same regions but taken from the upsampled patch using our algorithm.

stitching methods, the resolution is not fixed in advance in our approach, but we create the needed resolution on demand. The recursive warping and blending steps assure that details given by the input images appear at the correct positions and orientations in the output images, Figure 26c. For parts where no detail images are available, the synthesis can benefit from the derived knowledge of scale relations from the other input images. As shown in Figure 26d, plausible details can be added even for regions not covered by any of the detail images.

#### MULTISCALE TEXTURE SYNTHESIS

Using reflexive edges, i.e., loops, in the dependency graph allows us to produce results similar to a multiscale texture synthesis [107]. In Figure 27 a single input exemplar of size  $64 \times 64$  pixels was upsampled using our algorithm to  $2048 \times 2048$  pixels. Figure 27a shows zoom shots of the original input image, in Figure 27b our results are presented. Figure 28 shows a snapshot of the *Wheat field under dramatic sky* by Vincent Van Gogh whose resolution of  $256 \times 256$  was virtually increased to  $8192 \times 8192$  using a single exemplar with a single reflexive edge. While the main focus of our algorithm was not to create a new texture synthesis method, it is nevertheless applicable for this task as well.

#### ONLINE DATABASE

Our algorithm is also applicable to image collections from online databases. The result in Figure 29 used the first 35 hits on Flickr using the phrase *Big Ben*. The enlarged image in Figure 29a was then chosen manually to be augmented with details. Even

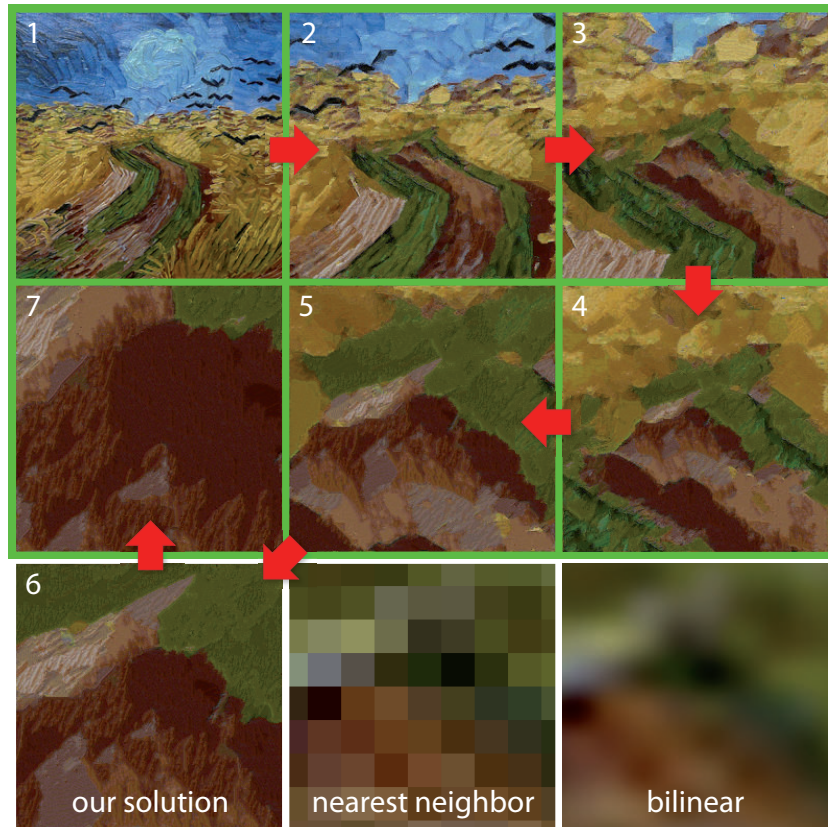


Figure 28.: Multiscale Texture Synthesis: The same  $256 \times 256$  pixel exemplar (upper left) is added to each level of the extended dependency graph facilitating an infinite zoom. Each subsequent image doubles the resolution, yet the quality remains high compared to bilinear or NN upsampling (bottom).

though the images have been taken by completely different cameras, angles, viewpoints and at different times, our algorithm adds plausible details to the root image. The algorithm found a substitute in the image database for the clock-face and replaced it. The other images were not used, because they differed too much.

Even for parts of the image where no detail information was available, plausible details have been added by our algorithm, see Figure 30. Our algorithm is no real super-resolution algorithm in the classical sense, as it provides only similarity to the input image but does not guarantee equivalence when down-sampling the result. On the downside this can lead to a minor loss of contrast in the image, as one pixel sized details might be replaced in the process. On the other hand this feature has several beneficial aspects. The algorithm has more freedom to add details to the image because of this loosened constraint and small artifacts, like the halo or compression artifacts around the tower are reduced in Figure 30c, also in the downsampled ver-



(a)



(b)



(c)

Figure 29.: Online Database: Our algorithm can derive relationships between photographs in image databases like Flickr [274]. These relations enable us to add specific details. (a) A subset of the first 35 images for the query *Big Ben* on Flickr. The enlarged image on the left was then chosen by the user. (b) Detail of the original image. (c) Result of our algorithm. A closeup on the small turret in the bottom right of the root image is given in Figure 30.

sion, Figure 30d. Classic super-resolution algorithms might even enhance these artifacts, Figure 30b.

The dependency graph of all examples in this chapter, except for Figure 23, 24, 27 and 28, have been automatically created by our algorithm.

#### 4.6 DISCUSSION

We have introduced a framework for detail enhancement in photographs. In this context, we presented a robust method to establish parent-child and scale relationships in unordered sets of photographs. Its derived graph structure enables us to find relations between images where simple feature matching would fail. We explained how to adjust the content of child images

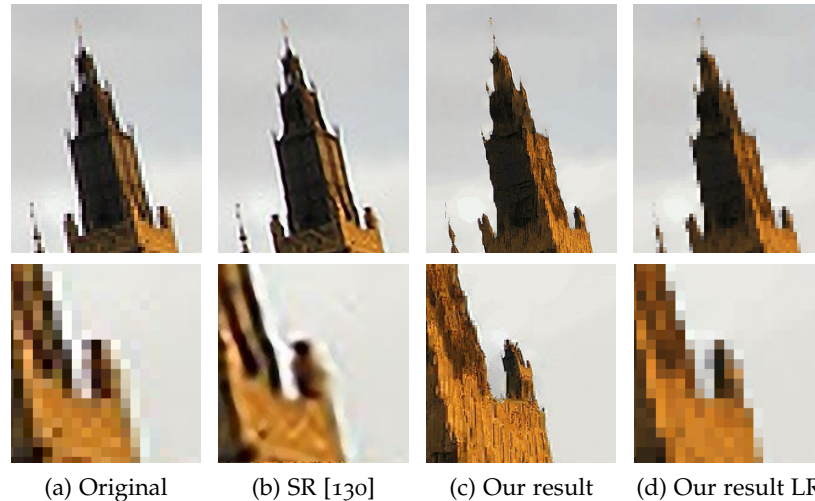


Figure 30.: Comparison to super-resolution in the *Big Ben* scene. Super-resolution approaches upsample images by guaranteeing equivalence to the original image after downsampling again. Our approach, instead, assures similarity to the original. This loosened constraint allows the algorithm to add new, plausible details to the image. (a) Original image. (b) Upsampled image by three octaves using the super-resolution algorithm by Kim *et al.* [130]. (c) Upsampled image using our proposed algorithm (d) Our result from (c) downsampled to the original size again. Note that none of the input images contained a close-up view of the turret.

to a user-selected image. This allows for a well-behaved detail synthesis and simplifies fusion. Additionally, we proposed an optimization-based approach for multiscale texture synthesis which adds details at various levels to the output image. It allows us to add specific details at specific positions. Combined with our dependency graph, even sub-pixel content with respect to the original image can be created. Our method is fully automatic, enabling novice users to create high-resolution results without a complicated or expensive setup.

#### LIMITATIONS

Despite feature matching and optimization steps, our method is still sensitive to parallax effects that occur if the images have been taken from different viewpoints. Splitting the input images into smaller patches and using voting (similar to [12]) might help to reduce these effects. It would also allow for a more flexible dependency graph, as overlaps in the images could be exploited more finely and the algorithm would rely less on the requirement of fitting homographies to entire images. On the other hand, smaller patches might reduce similarity to the original images, resulting in reduced overall quality.

Theoretically the SIFT features [158] we use for correspondence estimation are scale invariant. In practice, this is only true up to certain zoom factors. We cannot point out exact scaling differences between our input images. We could robustly estimate the homographies for an approximate scaling factor of up to 12. However, the factor may vary with the image content and is usually around two to three in our experience. To allow for more robust matching, we can make use of intermediate images taken from the according image pyramid.

The proposed structural adaptation in Section 4.3.2 works best if only few salient edges are to be matched. Due to the complexity of many natural images, robust automatic feature detection along the seam is still an open problem, as too many fine scale structures in the images can lead to erroneous matchings and false edge tracing.

In our current implementation, we wanted to be independent of reconstructed 3D geometry. But if enough images are available, it might turn out useful to incorporate 3D information as well, as it was done by Goesele *et al.* [94].

Our algorithm is also affected by strong changes in illumination of the different images. While the color correction step helps to resolve global color changes, it is currently not able to sufficiently remove artifacts caused by strong shadows. Working completely in the gradient domain and removing strong gradients, not in accordance to the parent image, or using intrinsic images similar to [155] might resolve this issue.

We currently assume that the detail images actually provide details. This is not the case if the objects of interest are out of focus. One might want to add an additional preprocessing step to remove such images.

#### FUTURE WORK

Currently, we have applied our algorithm only to relatively small databases consisting of a few dozen images. An interesting future direction might be to incorporate larger databases such as data from GPS or satellite views containing thousands or more images [112, 215]. In such a scenario, fast rejection and construction methods for the dependencies are needed. Using GIST [182] or scalable recognition and query approaches [86, 179] could speed up the process.

The time needed for detail synthesis can be quite extensive, as for each output pixel the best matching neighborhoods are to be found in each step. Faster optimization procedures like Instant Texture Synthesis by Numbers by Panareda Busto *et al.* [183] might reduce the overall computation time.

We would also like to give more artistic freedom to the user, e.g., marking regions with an interesting lighting condition to propagate this information to the rest of the image.

Investigating how to derive HDR information for the whole image if only details are captured with different exposure settings is also an interesting field for future research.



*All right everyone, line up alphabetically according to your height.*

— Casey Stengel

## 5.1 INTRODUCTION

In most interactive graphics applications, the scale at which some 3D object may be rendered during runtime is unknown beforehand. For small-scale depictions, well-known mipmaps avoid aliasing artifacts caused by texture minification [268]. On the other hand, if a textured 3D object ought to be displayed at a scale larger than the available texture map resolution, detail-deprived and washed-out renderings due to simple interpolation techniques are the result. We address the latter problem of texture magnification in this chapter.

In Chapter 4 we investigated a new approach to create high-resolution images from a set of low-resolution patches. Unfortunately, there are situations in which the creation of such a high resolution texture is unfeasible, e.g. in real-time rendering applications. Current commodity graphics hardware support only limited texture sizes, e.g.  $8192 \times 8192$  texels on an Nvidia GeForce 295 GTX. If larger textures are to be used, new techniques are needed that bypass this limitation. Instead of streaming large textures, which is a common practice [174], there are many situations in which it is sufficient to have high resolution

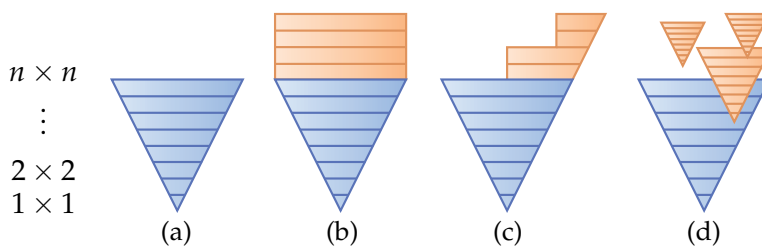


Figure 31.: Comparison between (a) standard mipmapping [268] – texture information is only provided up to a specific level; (b) clipmaps [238] – texture information is loaded on demand; (c) multiresolution textures [169] – a quadtree structure represents texture information at different levels; (d) our zipmaps [69] – a sparse representation to texture specific details at high resolution.

only for certain, specific and interesting parts in the texture. A zoom-into-parts texture map (zipmap), the technique we present in this chapter, enables rendering detailed close-up views of specific texture regions. In contrast to recent approaches like Gigapixel images [131] or clipmaps [238], we do not use a complete high-resolution texture map; instead, high-resolution texture insets are merged into low-resolution textures. In a nutshell, zipmaps can be thought of as a sparse representation of a larger mipmap, Figure 31. We show how zipmaps are almost as simple to use and render as standard texture mapping.

As particular contributions this chapter presents:

- a hierarchical texture mapping scheme, called zipmaps, which supports enhanced magnification of specific regions and naturally supports classic filtering techniques for anti-aliased rendering.
- a fast rendering algorithm for zipmaps, which enables applying zipmaps to arbitrary meshes in a single rendering pass.

The remainder of this chapter is organized as follows. We introduce our zipmap textures in Section 5.2 and show how they are applied and rendered. Section 5.3 presents resulting zipmaps in detail before we discuss limitations and conclude in Section 5.4.

## 5.2 ZIPMAPS

Zipmap textures can be thought of as a sparse sample representation of a large mipmap with almost arbitrary resolution. Up to a specific level  $n$  the whole texture pyramid is saved in a base level mipmap texture, called the *root*. This way standard minification methods can be used to prevent aliasing in cases where the texels projected into image space are smaller than a single pixel. To incorporate details for specific regions during magnification, additional texture pyramids, called children, are added at specific positions, if needed in a recursive manner. Hence, each one is associated with a unique texture matrix  $\mathbf{M}_i$  which transforms texture coordinates from the root to the  $i$ -th child patch for lookup.

The root also serves as a base layer for texture placement, i.e., the parameterization to establish the mesh/texture correspondence. The mesh, and therefore the designer, does not need to know anything about the placement of the detail texture patches as this is implicitly saved in the according texture matrices  $\mathbf{M}_i$ . Note that the base levels of these additional texture pyramids do not necessarily need to be at the highest level of the lower resolution parent image pyramid, nor do they need to be aligned

with any artificial structure, as it would be the case in quadtree or octree representations, Figure 31. The affected portions of the parent patch will be hidden behind the opaque regions of the detail patches. This enables a more efficient and flexible rendering.

For rendering, the root and children are assembled into a collection of ordered texture patches. Essentially, a zipmap texture is a simple collection of texture patches which are rendered in a specific order to texture an arbitrary surface. Patches containing the coarse overall information are rendered first, while child patches containing details are drawn later, on top of their parents. The ordering can be either set by hand or automatic techniques as presented in Chapter 4 can be employed.

The following is a description of the complete algorithm for rendering zipmaps onto arbitrary meshes. An overview of the complete process is also given in Figure 32.

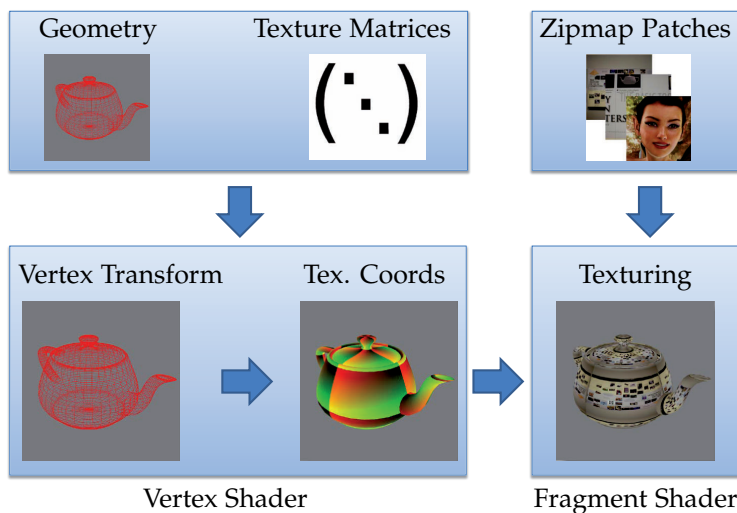


Figure 32.: Overview of the zipmap rendering technique: Applying zipmaps is almost as simple as plain texture mapping. The incoming texture coordinates are simply multiplied with the zipmap texture matrices and can then be used in the fragment shader directly for texturing.

### 5.2.1 Basic Rendering Algorithm

Rendering of zipmaps makes strong use of the traditional graphics pipeline for efficiency. Broadly speaking the classic programmable graphics pipeline can be divided into two main parts, a vertex and a fragment shader. The main purpose of the vertex shader is to transform the model vertices and its associated attributes, like texture coordinates. These are interpolated and passed to the fragment shader, which computes the final output

color. This simplified description of the pipeline shall be sufficient for our purposes.

In the vertex shader the texture coordinates for the root patch are queried from the vertex attributes. Multiplication with the matrices  $\mathbf{M}_i$  results in the corresponding texture coordinates for each child patch  $i$ . This transforms the texture coordinate from the root patch's coordinate system into the child coordinate system.

A simple texture lookup in the fragment shader then fetches the corresponding value for the needed output pixel. We compute the final color value of the  $\text{rgb}\alpha$ -quadruple  $C$  by combining all texel  $\text{rgb}\alpha$ -values using Equation (5.1).

$$C = \sum_i \omega_i C_i , \quad (5.1)$$

where

$$\omega_i = \alpha_i \prod_{j>i} (1 - \alpha_j) , \quad (5.2)$$

i.e. we simply mix the color value  $C_i$  of a patch with the already computed color according to the alpha channel of the patch. So in most cases a new patch is simply drawn over the old one, as most parts of the texture patches are opaque. We will elaborate on this fact further in Section 5.2.3. In order to prevent drawing child patches if the calculated texture coordinates are outside the  $[0 \dots 1]$  range we can make use of hardware texture clamping, see below.

### 5.2.2 Extended Rendering Algorithm

If a zipmap consists of more patches than the GPU supports in a single rendering path, we use a slight variant of the aforementioned strategy. In a first pass, the first  $m$  patches are drawn and written to the framebuffer as described before.  $m$  is the maximum number of possible patches to be rendered in a single pass due the hardware limitations of available texture units. Using multiple render targets, we also render the current texture coordinates of the root patch into the red and green channel of another buffer  $\mathbf{B}_{tc}$  which is initialized to zero beforehand, and set the alpha value to one, to mark affected fragments. In the next pass, we bind the next texture patches to the texture units plus the buffer  $\mathbf{B}_{tc}$  containing the texture coordinates. Now instead of rendering the whole textured mesh again, we simply draw a screen filling quad and calculate the texture coordinates of the children in the fragment shader by making use of  $\mathbf{B}_{tc}$ . If its alpha value is zero, we discard the fragment, keeping the old color value. Otherwise we multiply every  $\mathbf{M}_i$ ,  $i > m$ , with the

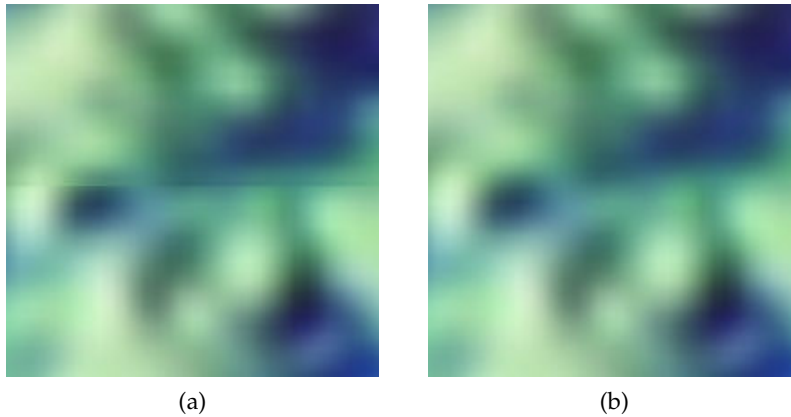


Figure 33.: Error-prone hardware-based texture filtering: (a) Close-up view with artifacts at patch borders (horizontal line in image middle). These appear even if the actual texel values are the same for the patch and the background. (b) Setting the alpha value to zero at patch boundaries for every mipmap level removes the seams.

queried texture coordinate from  $\mathbf{B}_{tc}$  to calculate the correct texture coordinate for the  $i$ -th patch and color the output fragment as described in 5.2.1. We can repeat this process until every texture patch has been processed. Texture instancing, i.e., if the same detail patch should appear more than once on the surface, can be achieved by using different texture matrices for the same patch.

### 5.2.3 *Blending Patches*

Current graphics hardware poses another problem whenever texture patches are drawn on top of each other. If texture values close to a patch boundary are queried, hardware interpolation will not always be able to query the correct texture value, which will create a seamless blending with the background. This problem persists, even if exactly the same colors are used for both patches. This is due to the employed hardware interpolation methods for border conditions which causes visible seams, Figure 33a. This problem can be solved by setting the alpha-channel at the border of zipmap patches to zero, Figure 33b. This is done for every level of the mipmap pyramids during the zipmap generation process. Another advantage of this approach is that patches becoming smaller than one pixel in the output image simply disappear and do not produce small pixel artifacts that would otherwise be visible. In addition, if seamless merging of the whole content is needed, we use the approach presented in Chapter 4 to create the corresponding alpha maps.

### 5.2.4 Repeating, Clamping and Mirroring

Graphics APIs like OpenGL allow to assign texture coordinates other than the  $[0, 1]$  range to a model's vertices. This is useful to create copies of the same texture on the objects surface. In the classic pipeline the texture coordinates are projected back into the  $[0, 1]$  range right before the texture lookup in the fragment shader. The above zipmap procedure however requires the projection already in the vertex shader in order to compute valid texture coordinates for the child patches. This can be achieved with a small adaptation in the vertex shader. What is needed is the correct root's texture coordinate in the  $[0, 1]$  range before multiplication with any  $\mathbf{M}_i$ . If the roots texture coordinates are set to clamp (`GL_CLAMP` in OpenGL), we clamp them to the  $[0, 1]$  range. For repeating (`GL_REPEAT`) only the fractional part is needed. For mirroring (`GL_MIRRORED_REPEAT`) we need to use the following code snippet.

```
t = |x| % 2.0;
return (t < 1.0) ? t : (2.0 - t);
```

where  $x$  is the roots texture coordinate and  $t$  is the roots coordinate transformed to the  $[0, 1]$  range.

## 5.3 RESULTS

Zipmaps can be easily rendered in real-time, since for each patch only a single matrix multiplication per vertex and one texture lookup per fragment are required. The memory requirements are in direct accordance to the number and size of the input images used. No additional information other than the patches and their texture matrices need to be saved. Since the child patches are saved in relation to the root patch, the application programmer only has to define texture coordinates for the root patch, just as he would do with a conventional 2D texture, making the zipmaps very easy to use in practice.

As test data, we have taken input images with a handheld camera. To automatically align and adjust the images we used the method described in Chapter 4. Figures 34 to 36 show results of zipmap rendering.

On the top left of each image, the input patches are shown. On the right the zipmap texture is applied to different geometries, and some close-up views from different viewpoints and different distances are shown. The output screen resolution was always set to  $1024 \times 1024$  pixels, so magnification is present in most views. Our zipmap textures can be easily applied to any kind of geometry. In Figure 34 we use a four patch zipmap to texture a teapot. In Figure 35 and 36 we apply a zipmap consisting of

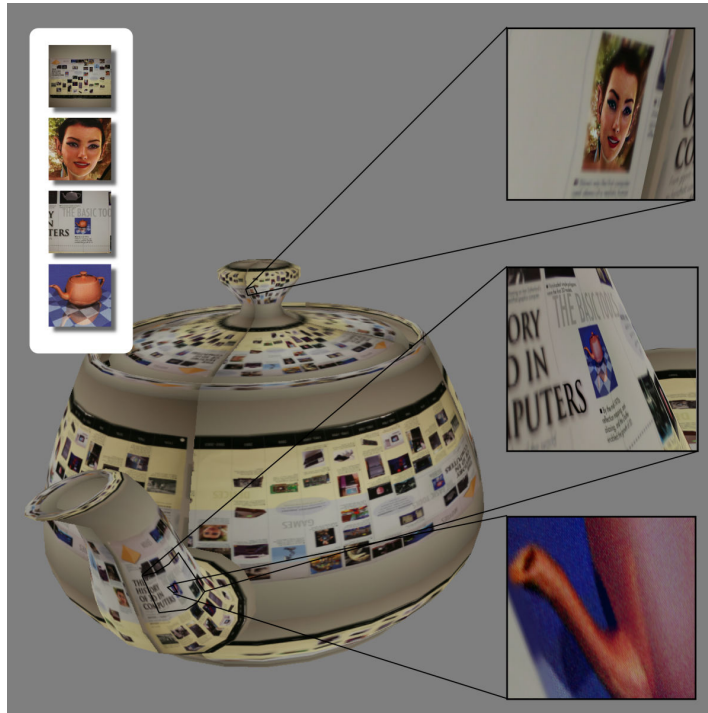


Figure 34.: Zipmap textures can be easily applied to any geometry, just like conventional textures. Four input images of sizes between  $428 \times 428$  pixels and  $512 \times 512$  pixels are used to create a virtual resolution of 1.2 gigapixels in specific regions.

four patches and six patches, respectively, to a simple quad for illustration purposes. Zooming onto single droplets or the knot-hole is now possible. Figure 35 shows an interesting showcase example. Due to the large depth conveyed in the scene and the accompanying strong parallax effects, some small ghosting artifacts are visible. Interestingly, even though the water fountain changes quite a lot during the acquisition no temporal artifacts are visible because the detail resolution patch is always drawn on top of the low resolution images.

#### 5.4 DISCUSSION

We have introduced the concept of zipmaps, a method for rendering detailed close-up views of textured surfaces. Zipmaps are an easy to use, flexible and simple representation for multiscale texture maps, as many in-built functions of the graphics hardware are exploited. Zipmaps exploit graphics hardware filtering capabilities to produce anti-aliased results. They can be used with arbitrary images and different kinds of textures; even normal or displacement maps could be processed.

In comparison, a typical approach in the games industry is to render detail textures as textured detail geometry. While per-



Figure 35.: Zipmap of a facade with fountain. Time-varying parts of the scene are merged into a common representation. Four input images of size  $512 \times 512$  pixels are used to create a virtual resolution of 20 megapixels in the central portion of the scene.



Figure 36.: A zipmap texture acquired from six photographs of size  $512 \times 512$  pixels each and applied to a simple quad to create a virtual resolution of four gigapixels in the depicted area.



forming an optimal amount of per-pixel work, this approach has the drawback of z-fighting if the detailed geometry is too close to the original. Visible seams appear if the border handling is not done correctly, or the viewpoint gets too close to the surface. To prevent these effects the geometry is usually cut into several non-overlapping pieces, which is time-consuming and requires a lot of manual work compared to our approach.

#### LIMITATIONS

For the results presented in this chapter only a few patches are used per zipmap, therefore performance is of no concern. With increasing patch number the computational effort as well as the memory consumption will increase linearly. One might argue that zipmaps perform a lot of texture accesses if the number of zipmap patches is large. While this is true, it is not as crucial as it might sound at first. If the patch is outside the viewing frustum the according texture coordinate will usually be clamped to the same value for most or all pixels in the output image. Therefore access becomes cheaper, because the texture region will be cached by the graphics hardware. This is more effective than using a conditional statement to query for a valid texture coordinate.

If the detail patches cover a large area of their parent patches some memory is wasted. In these cases it might be more beneficial to merge the patches into a single texture map, but then flexibility is lost.

For future work it would be interesting to investigate the use of multiple indirection textures that are updated on demand to reduce the texture access and memory overhead. Animated zipmap textures would enable us to create other interesting effects, as our concept allows direct integration of moving patches.



Part III.

## Error Concealment in Video Matting



*It was the silhouette.*

— John Galliano

In Chapter 4 we introduced a system to upsample and seamlessly merge images into a common image domain. However, the content in these merged images depicted basically the same scene. Therefore, all objects could be treated as opaque, and the main challenge was to adjust the images properly to make merging into a single image domain feasible. If more complex objects are to be composed, e.g., semi-transparent objects, like animals or humans with fur or hair, the problem changes. In this case not only the composition of the source and target is needed, but the source needs to be extracted from its surrounding image content. This process is generally called digital matting, or digital video matting if the foreground is to be extracted for a complete video [253]. In this part of the thesis we propose a new matting algorithm for videos containing complex objects.

## 6.1 BACKGROUND

In digital matting a foreground object along with an opacity estimate for each pixel is extracted. This opens up the possibility to seamlessly insert new elements into the scene, e.g., an actor can be recorded and later pasted into a different scene. Such techniques are frequently used in commercial television or film production [34]. Hence, the alpha matte as well as fore- and background needs to be estimated for an entire video sequence. In addition, the extracted alpha mattes may also serve in scene geometry reconstruction, e.g. in multiview setups, cf. Chapter 2.8.

For controlled environments, like a blue screen studio [218], or if the background is known [186] the problem of digital matting is considered to be solved. In this case the extraction may also be performed on a per-frame basis for complete video sequences. But the problem becomes more complicated when loosening the constraints. If the background is unknown, the problem of foreground estimation becomes ill-posed. Additional user-preset constraints on fore- and background are needed, as there are only three knowns, the *rgb* image pixel values, and

seven unknowns, the fore- and background colors plus the corresponding alpha values of the pixels. Since not only a binary segmentation into fore- and background is needed, complete alpha mattes must be estimated. This problem is also called *natural image matting* [145]. If we extend the problem to video, robust techniques are required, to automatically pull a good matte for several frames. It turns out that few algorithms manage to do this robustly for more than a dozen frames without additional help from the user [46] or additional hardware [128, 170, 171]. If the problem is extended to multiview recordings the number of frames that have to be processed increase tremendously, emphasizing the need for techniques which are simple, efficient and robust. We will present such an approach in Chapter 7.

## 6.2 RELATED WORK

The most simple matting technique is arguably *blue screen matting*, also known as *chroma keying* [218]. Foreground elements are recorded in front of a solid color background, and a number of heuristics are used to extract the matte for each frame. While being fairly effective, the method requires a controlled studio environment.

More sophisticated methods such as natural image matting do not impose such strong restrictions on the background. However, the problem becomes inherently under-constrained and additional information in form of a trimap [45, 100, 231], fore- and background scribbles [103, 145, 252] or tracing along the edges between fore- and background [255] is required.

Our work is mostly inspired by the *spectral matting* approach by Levin *et al.* [144]. Spectral matting extends the ideas of spectral segmentation [178, 262, 277]. The real-valued matting components are obtained via a linear transformation of the smallest eigenvectors of the matting Laplacian matrix [145]. These matting components are then combined to form the complete foreground matte.

While object cutout in still images has more or less been solved, video matting remains a challenging problem. In video matting the task is to estimate the foreground matte for each frame of a video sequence with a minimal amount of additional manual editing. A similar approach to blue screen matting is *difference matting* [129] where the mapping of the difference between the recorded scene and a background shot yields the opacity values. In *rotoscoping* a user draws an editable curve, like a B-spline, around the foreground element at selected keyframes, often with the aid of snapping tools that are auto-aligned along high gradient areas [6, 28, 93, 175]. These are then interpolated between the keyframes. However, a lot of manual adjustment is required

to pull a high-quality matte, and the matte is only binary and usually does not provide any alpha values for blending.

Different directions have been explored recently in the field of video matting. Graph cut segmentation has been extended to work directly on the 3D video volume [15, 148, 254] and spatially varying color models have been tracked [278]. The recently published *Video SnapCut* approach by Bai *et al.* [17] combines a set of local classifiers with a coherent matting approach to achieve high-quality results with the possibility for local refinements. Still it requires considerable user-interaction for longer sequences.

A common technique to guide a segmentation result over time is to use optical flow [17, 28, 46]. Unfortunately optical flow can introduce small errors which accumulate over time and diminishes reliability of the estimation, forcing user intervention to guide the algorithm. Matting techniques presented in the literature based on flow propagation reported that typically the alpha matte for up to a dozen frames can be pulled on average without user interaction, [46, 203]. In contrast, our approach, introduced in Chapter 7, reinitializes the foreground estimation on a per-frame basis, enabling more robust propagation. In the field of multiview matting, Sarim *et al.* [203] proposed a method for trimap propagation and refinement in sparse multiview setups. As their approach models fore- and background by color statistics it is independent of the baseline between cameras allowing for very wide baselines up to  $180^\circ$ . As with any purely color-based matting algorithm, the technique of Sarim *et al.* works very robust (up to several hundred frames) if the background model differs enough in comparison to the foreground model.

One persistent, common problem is often the slow speed of matte computation. Only recently the first real-time matting approaches have been proposed [90, 97]. But both still rely on a trimap to initialize the algorithms for each frame, leading to potential problems if the trimap is not propagated sufficiently. Our approach is not real-time capable in its current version, but it still allows for interactive matting sessions and provides room for speed improvements, as we will explain.

A recent comprehensive survey on different matting techniques can be found in [253].





*Computers are useless. They can only give you answers.*

— Pablo Picasso

## 7.1 INTRODUCTION

In this chapter we will concentrate on extracting the alpha mattes from videos so that they can be used for sparse multiview reconstruction techniques or digital matting. Our goal is to provide a video matting technique that complies with the following requirements:

- Complex objects
- Speed
- Robustness
- Simplicity
- Intuitive behavior

By complex objects we mean that a complete alpha matte including transparency information is extracted. The approach should be fast enough to allow for interactive matting sessions. Very few user interaction should be needed. If user interaction is required, it should neither require much time nor much knowledge on how to perform the correction. By intuitive behavior we mean that the algorithm should support a natural workflow, allowing the user to edit a video stream by passing over it only once. Many video matting algorithms work in a forward-backward manner, i.e., information, like user-interaction, is propagated in both directions forward and backward in time. While this might be beneficial to reduce the amount of required user interaction, it also complicates the interaction. The user needs to jump forward and backward in time randomly accessing and correcting the video. So the user hardly knows when he is really finished without checking the whole video again. We believe that a robust forward scheme is beneficial for a more intuitive behavior of the matting algorithm.

The rest of this chapter is organized as follows. In Section 7.2 we give an introduction to spectral matting for still images and

point out the benefits of adapting this approach for video matting. In Section 7.3 we present our extended approach for video matting, including initialization, propagation and optimization of the alpha mattes for each frame of a video sequence. We present and discuss results for different test sequences in Section 7.4 and 7.5.

## 7.2 SPECTRAL MATTING

The *compositing equation* (7.1) describes the digital matting process as a linear combination of foreground color  $F$  and background color  $B$  in every pixel:

$$\mathbf{I}(x, y) = \alpha(x, y)\mathbf{I}_F(x, y) + (1 - \alpha(x, y))\mathbf{I}_B(x, y), \quad (7.1)$$

where  $\mathbf{I}(x, y)$  is the pixel color at position  $(x, y)$  and  $\alpha(x, y) \in [0, 1]$  is the alpha matte value at position  $(x, y)$ .

Spectral matting by Levin *et al.* [144] generalizes this idea to multiple layers:

$$\mathbf{I}(x, y) = \sum_{k=1}^K \alpha_k(x, y)\mathbf{I}_{F_k}(x, y), \quad (7.2)$$

with  $\sum_{k=1}^K \alpha_k(x, y) = 1$  and  $\alpha_k(x, y) \in [0, 1]$ . Where  $K$  is the number of layers,  $\mathbf{I}_{F_k}$  and  $\alpha_k$  are the different matting components encoding the influence of  $\mathbf{I}_{F_k}$  at each pixel  $i$ . Despite its proven high-quality (see [144] for a comparison to other methods) the real benefit of spectral matting for video matting is the decomposition into  $K$  alpha matting components  $\alpha_k$ . All that is needed to obtain the desired foreground is to specify the components belonging to this. Suppose  $\alpha_{k_1} \dots \alpha_{k_n}$  are designated as belonging to the foreground, then the complete matte  $\mathbf{M}$  is obtained by simply adding them together  $\mathbf{M} = \alpha_{k_1} + \dots + \alpha_{k_n}$ . An example is given in Figure 37, where the red marked components are added together for the final foreground estimation.

### 7.2.1 Benefits of Matting Components

Locally grouping pixels of similar attributes into larger but still comparatively small agglomerates, also called clusters, matting components or superpixels, leads in general to more robust results when used as computational atoms in comparison to single pixels [79, 197]. Using matting components overcomes several disadvantages of the classic trimap, where all pixels are marked as foreground, background and unknown, which is one of the most common techniques to initialize matting algorithms [14, 45, 46, 231]. The main drawbacks of those algorithms relying on trimaps for initialization is their inability to correct imprecise

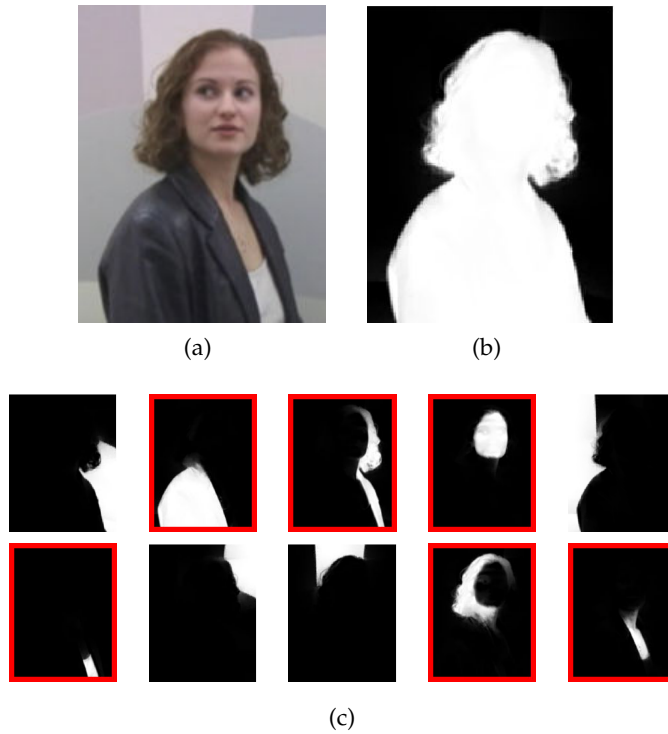


Figure 37.: Matting Components: (a) The input image. (b) The estimated alpha matte  $M$ . (c) The matting components  $\alpha_k$ . Components of the foreground are marked in red.

maps. Once a pixel is considered as fore- or background its truth value is no longer questioned. Therefore precise trimaps are a necessity for those algorithms.

Another drawback is that the alpha values in the unknown regions are usually derived from the estimated local color models of the known regions. If the band of unknown pixels is too large, the algorithms will fail as the assumption of correct color models from the surrounding known pixels is no longer true.

Another classic approach in video matting is to use a two-step algorithm. First a hard segmentation is computed, e.g. by translating simple user-specified constraints into a min-cut problem, which can be solved optimally using graph-cuts [148, 201, 254]. The hard segmentation could be transformed into a trimap by morphological operators such as dilation and erosion [253]. But as the amount of erosion and dilation is based on a user-specified value or a global constant some fine or fuzzy features might be missed. This problem becomes even more problematic if the trimap is propagated over several frames.

### 7.3 SPECTRAL VIDEO MATTING

In the following our approach is presented, which makes use of the characteristics of spectral matting [144], and extends it to video matting with minimal user input.

#### 7.3.1 Initialization

In a first stage, the matting components for the complete video sequence are extracted using the spectral matting technique [144] and a user-specified value for  $K$ , i.e., the number of estimated clusters, ten worked well for our test scenes. For cutting out certain foreground objects out of a multitude of possible objects the user has to decide which components  $\alpha_k$  should be part of the foreground. The sum of these  $\alpha_k$  form the matte  $\mathbf{M}_0$  for the first frame. As we chose a small number of clusters for our test scenes, the foreground clusters can be chosen in a manner of seconds in most cases. For some simpler scenes even this initial foreground estimation can be automated by using unsupervised matting as described in [144], providing the user with an initial guess of what could be the foreground.

A simple scribble interface could be incorporated for selection, if more clusters are needed for a complex scene. The user can draw simple strokes into the first frame and for all scribbled pixels the cluster with the largest opacity value at that position is added to the solution. The best performance is achieved if the number of clusters is as small as possible, still fulfilling the constraint that two distinct objects do not share a cluster, because otherwise a separation would be impossible. Apart from that, this step is the only user interaction that is needed for the algorithm to start.

#### 7.3.2 Propagation

Using this initial set of matting components, the foreground is propagated through the video volume as presented in the following. Given two neighboring frames  $\mathbf{I}_{(j-1)}$  and  $\mathbf{I}_j$  plus the matte  $\mathbf{M}_{(j-1)}$  for the  $(j-1)$ -th frame, we can compute a relation between  $\mathbf{I}_{(j-1)}$  and  $\mathbf{I}_j$  to satisfy the following equation:

$$\mathbf{I}_j \approx \mathbf{W}_{\mathbf{I}_{(j-1)} \rightarrow \mathbf{I}_j} \circ \mathbf{I}_{(j-1)} \quad , \quad (7.3)$$

where  $\mathbf{W}_{\mathbf{I}_{(j-1)} \rightarrow \mathbf{I}_j} \circ \mathbf{I}_{(j-1)}$  warps an image  $\mathbf{I}_{(j-1)}$  towards  $\mathbf{I}_j$  according to the warp field  $\mathbf{W}_{\mathbf{I}_{(j-1)} \rightarrow \mathbf{I}_j}$ . The problem of determining this warp field  $\mathbf{W}_{\mathbf{I}_{(j-1)} \rightarrow \mathbf{I}_j}$  is known as optical flow or correspondence estimation. In our case we are not interested in warping the image itself to the next frame but rather the alpha matte.

We compute an initial guidance  $\mathbf{G}_j$  for the foreground matte of frame  $\mathbf{I}_j$  by warping  $\mathbf{M}_{(j-1)}$  using the warp field  $\mathbf{W}_{\mathbf{I}_{(j-1)} \rightarrow \mathbf{I}_j}$ :

$$\mathbf{G}_j = \mathbf{W}_{\mathbf{I}_{(j-1)} \rightarrow \mathbf{I}_j} \circ \mathbf{M}_{(j-1)} \quad (7.4)$$

This guidance matte cannot reveal a precise alpha matte, as, correspondence estimation algorithms do not incorporate transparency into their motion model, in general. But we can still use it as an initialization to estimate an optimized alpha matte for frame  $\mathbf{I}_j$ , Figure 38 second row.

### 7.3.3 Matte Optimization

To estimate the foreground matte  $\mathbf{M}_j$  of frame  $\mathbf{I}_j$  we search for the combination of precomputed foreground clusters  $\alpha_k$  which minimizes the difference between the resulting new alpha matte  $\mathbf{M}_j$  and the initial estimate  $\mathbf{G}_j$  in a least-squares sense. This reconstructs the lost fine details and removes the error introduced in the warping procedure. Therefore the task is to minimize the following error function:

$$E = \sum_{(x,y) \in \Omega} \left\| \mathbf{G}(x,y) - \sum_{k=1}^K b_k \alpha_k(x,y) \right\|, \quad b_k \in \{0,1\} \quad (7.5)$$

where  $b_k$  is the binary solution vector we solve for.

Unfortunately this problem seems to be NP-hard and exponential in the number of clusters. But we can restrict the number of clusters by removing those from the set which would definitely increase the error, i.e., those for which

$$\sum_{(x,y) \in \Omega} \left\| \mathbf{G}(x,y) - \alpha_k(x,y) \right\| > \sum_{(x,y) \in \Omega} \left\| \mathbf{G}(x,y) \right\| \quad (7.6)$$

As it would still be unfeasible to exhaustively compute all possible combinations we use a greedy approach which solves this optimization problem well in all our encountered test cases. Starting with an empty initial estimate for  $\mathbf{M}_j$ , we assume there is no foreground in the image and all  $b_k$  are 0. We then add the single matting component  $\alpha_k$  to our solution which reduces the error function the most and set the appropriate  $b_k$  to 1. The process is repeated until  $E$  converges to a minimum.

As the clusters for the new image have been computed beforehand and are independent of the solution of the previous frame, this method works robust even in the case where the optical flow cannot compute precise warp fields, see Figure 38 for a comparison. Also disocclusion, i.e. newly appearing regions, which can be a difficult problem, are handled robustly if the disoccluded parts belong to the same cluster as already visible parts, e.g. an object turning revealing new visible parts each frame.

Repeating the described process of warping the previous matte to the current frame and reestimating the foreground for the following frames computes the alpha matte for the whole video sequence. The computation of the optical flow could also be computed in the preprocessing stage, but as fast optical flow implementations exist, this would only waste storage space. To prove the inherent robustness due to the reinitialization of the foreground using the matting components, we used a simple block-matching method [119], being aware that better optical flows exist, which could be incorporated in future versions.

#### 7.3.4 Keyframe Editing

In some situations, it is possible that certain changes of the shape of the foreground object cannot be identified automatically. This situation occurs because optical flow algorithms are only suitable for small and smooth motions, in general, and not for strong changes in the shape of the foreground. Most of the times the re-estimation using the matting components handles also imprecise flows, but if the error becomes too large manual adjustment of the user is necessary. In this case, the user scans the results until he finds a frame which has an incorrectly estimated foreground. He then adds a new keyframe by reinitializing the foreground clusters manually, as done for the first frame. The algorithm then recomputes the rest of the video with the new foreground estimation. If the content of a scene changes drastically throughout the video, it might be helpful to not only reinitialize the foreground clusters, but also to change the number of cluster.

## 7.4 RESULTS

Our test PC used for the evaluation is equipped with an Intel Core2Duo with 2.40GHz, only one core used, with 2GB of RAM. As test scenes we used two commonly used test sequences kindly provided by Y.Y. Chuang [46], namely Amira, Figure 37, and Kim, Figure 38. Both contain complex foreground shapes and motion, like hair, plus a non-static background, and occlusion and disocclusion is apparent.

In a pre-processing phase the spectral clusters are computed and saved to disk, which, using the Matlab implementation of Levin *et al.* [144], takes approximately 5 minutes for the computation of a  $360 \times 240$  frame. For the interactive online phase, specifying the foreground clusters took less than 10 seconds for a trained user while editing further keyframes takes even less time. Therefore the user interaction involved in matting each of our test scenes took less than one minute for the complete Amira video, consisting of 66 frames, Figure 37. The Kim sequence, con-

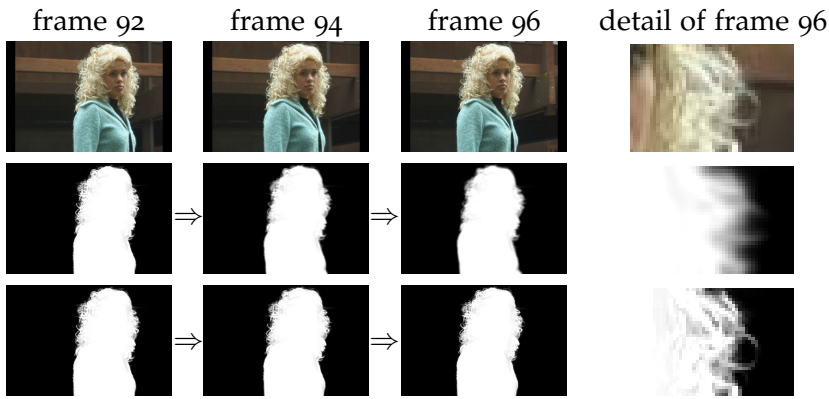


Figure 38.: Propagation error: The potential to reinitialize the warped alpha matte in our approach improves the final result in alpha matte estimation for successive frames and prevents accumulated errors. For the presented sequence (shown above) the alpha matte is provided for frame 92. Warping the alpha matte in forward direction yields mattes with an increasing error (second row). Reinitializing the alpha mattes with our proposed technique shows improved results without visible accumulated errors (bottom row).

sisting of 102 frames was computed completely automatic without any user interaction at all, Figure 38. The initial alpha matte was automatically estimated using the best hypothesis from the unsupervised matting [144].

The computation of the optical flow plus warping of the alpha matte and optimization for the next matte took less than one second per frame or milliseconds if we would use our GPU Optical Flow from Chapter 11. Generally the spectral matting performs very well even for complex structures such as hair. Very fast movements or newly appearing foreground objects usually require some user interaction, as the optical flow will fail in these cases. The number of edited frames, along with the computation times are given in Table 2. In the case a binary alpha matte is needed, e.g. for 3D reconstruction based on silhouettes, the result from the spectral video matting can be converted to a binary matte by a simple thresholding.

## 7.5 DISCUSSION

In this chapter, we have presented a new, easy to use technique for pulling foreground mattes with complex silhouettes from a video sequence. Our approach is based on different methods and combines their individual strengths. By combining spectral matting and optical flow, we obtain high-quality mattes for different recorded video scenes. We introduce a simple yet efficient way to propagate the alpha matte information to successive

sequence	resolution	frames	keyframes	components	fps
Amira	$360 \times 240$	66	1	10	1.23
Kim	$360 \times 240$	102	0	10	1.03

Table 2.: Details for the sequences used in our algorithm.

frames. We optimize the new foreground matte before propagating it to the next frame to prevent accumulated errors. We have shown that our approach can even pull a high-quality alpha matte from a complete video without any user interaction at all.

Current limitations of our approach are its high memory requirements and long preprocessing time, due to the spectral matting algorithm, and that we have not yet included more sophisticated user interaction techniques for cases when the spectral matting fails to estimate good clusters. This can also happen if the images are cluttered. This is a known drawback of the spectral matting technique [144, 253]. One way to diminish some of these errors is temporal filtering of the alpha mattes using the flow fields in order to find the temporal neighbors in successive frames. By comparing the result of different numbers of matting components during guidance map optimization it may be possible to find an optimal number of clusters automatically. A hierarchical subdivision scheme for local matting components would also be an interesting future research direction. Our matting prototype is currently implemented on the CPU, but porting the optimization and flow computation to the GPU is relatively straight-forward and should allow for real-time interaction. Background estimation as done in [46] could further improve the results.



Part IV.

# Error Concealment in Image-based Rendering



*It is the nature of all greatness not to be exact.*

— Edmund Burke

In the last two parts of this thesis we examined the problem of error concealment in image compositing tasks and video matting. In this part we will lift the problem to the third dimension and focus on the error sources and ways to deal with visible rendering artifacts in image-based rendering systems and free-viewpoint video, cf. Section 2.9. After a short introduction into the topic and related work, we will begin in Chapter 9 with a more formal analysis on what causes these rendering artifacts. Based on this analysis we are able to derive new rendering strategies which will be presented in Chapters 10 and 11.

## 8.1 BACKGROUND

To take advantage of the continuing progress in graphics hardware capabilities for realistic rendering, ever more detailed model descriptions are needed. Because creating complex models with conventional modeling and animation tools is a time-consuming and expensive process, direct modeling techniques from real-world examples are an attractive alternative. By scanning, or reconstructing, the 3D geometry of an object or scene and capturing its visual appearance using photos or video, the goal of direct modeling techniques is to achieve photo-realistic 3D rendering results at interactive frame rates.

Texture mapping was introduced in computer graphics as early as 1974 as a very effective means to increase visual rendering complexity without the need to increase geometry details [41]. Later on, projective texture mapping overcame the need for explicit texture-to-surface parameterization and enabled applying conventional photographs directly as texture [206]. To texture 3D geometry from all around and to reproduce non-Lambertian reflectance effects, View-dependent Texture Mapping [59] and Unstructured Lumigraph Rendering [36] blend multiple photographs taken from different viewpoints on the object's surface or, in the output image domain, respectively. If exact 3D geometry and sufficiently many, well-calibrated and registered im-

ages are available, image-based modeling techniques [142, 270] achieve accurate and photorealistic 3D rendering results.

Unfortunately, acquiring highly accurate 3D geometry and calibrated images turns out to be at least as tedious and time-consuming as model creation using software tools. In response, a number of different image-based rendering techniques have been devised that make do with more approximate geometry or no geometry at all.

Pure image-based techniques that do not use any geometry proxy have the advantage that instead of reconstructing the scene explicitly, only correspondences between input images are estimated. From a local perspective there is no difference between a moving camera and a moving scene, therefore time and space can be treated equally to allow for space-time interpolation [76, 227, 228, 229]. If combined with sophisticated image interpolation techniques, high-quality results can be achieved [104, 151, 161]. A drawback of these approaches is that they can only interpolate between images, i.e., the virtual camera is bound to move on the manifold that is spanned by the input cameras. An exception is the work of Einarsson *et al.* [66] where the only restriction is that each viewing ray has to intersect this manifold. Another drawback of interpolation techniques is that they require very precise correspondence fields between the images. Classic optical flow techniques usually fail if the disparity between the images is too large. While techniques for long-range correspondence estimation exist [151, 154, 202, 224, 225], they usually infer very high computational costs.

In this part of the thesis we focus on image-based rendering techniques using geometry proxies. With a mere planar rectangle as geometry proxy, Light Field Rendering arguably constitutes the most “puristic” geometry-aided image-based rendering technique [146]. Here, many images are needed to avoid blurring or ghosting artifacts [42]. If more appropriate depth maps are additionally available, Lumigraph rendering can compensate for parallax between images to yield convincing results from less images [36, 99]. Other image-based rendering approaches implicitly or explicitly recover approximate 3D geometry from the input images to which the images are applied as texture [40, 166, 251]. In general, however, the price for contending with approximate 3D geometry is that (many) more input images must be available, otherwise rendering quality degrades and artifacts quickly prevail, see Figure 39. We will present a thorough analysis of these artifacts in Chapter 9.

Years of research have shown that determining stereo correspondences for 3D reconstruction is a difficult problem. In general, many current methods require the images to be of similar appearance, as is the case with human binocular vision. If

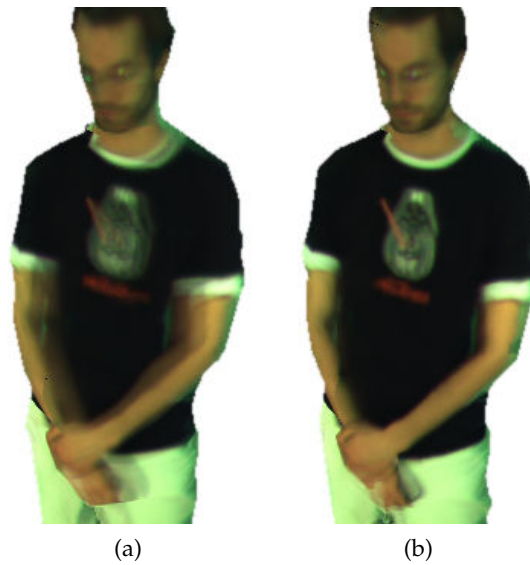


Figure 39.: Comparison of (a) standard linear interpolation using the Unstructured Lumigraph weighting scheme [36] and (b) our Floating Textures approach, Chapter 11, for similar input images and a visual hull geometry proxy. Ghosting along the collar and blurring of the shirt's front, noticeable in linear interpolation, are eliminated on-the-fly by Floating Textures [72].

the distance between the cameras, also called baseline, increases, precise reconstruction becomes difficult due to different degrees of foreshortening, occlusion effects and large disparities, all of which aggravate the reconstruction. The alternative of smaller baselines has the disadvantage that computing depth becomes very sensitive to noise in image measurements, and more cameras are needed again for a full reconstruction of the scene.

Another unsolved challenge is that, while image-based rendering can compensate for coarse 3D geometry if sufficiently many input images are available, all techniques still require well calibrated cameras. Thus, time-consuming camera calibration procedures must precede image-based object acquisition. While techniques exist to speed up the calibration process [152], they come at the cost of less precise calibration. But even if utmost care has been taken during acquisition, minute camera calibration inaccuracies, tiny 3D scanning holes, and small registration errors can visibly degrade rendering quality of the model. The only option to rescue rendering quality then is to try to "fix" the model, registration, or calibration by hand, or to repeat the acquisition process all over again. What is needed is a multi-image texturing algorithm that achieves best-possible results from imprecisely calibrated images and approximate 3D geometry.

In the next chapter we start with an analysis of how these deficiencies influence rendering quality of image-based free-viewpoint systems. In Chapter 10 and 11 we present algorithms to deal with these problems.

## 8.2 RELATED WORK

### SAMPLING PROBLEM

In light field rendering, a novel view is generated by appropriately re-sampling from a large set of images [146, 165]. Given sufficiently many input images, the synthesized novel view can be of photo-realistic quality. Else, ghosting artifacts or, at best, blurring degrade light field rendering quality. Several researchers have investigated how many input images, more precisely samples, are minimally needed to create artifact-free light field rendering results [42, 149, 150]. By employing a prefiltering step the number of necessary samples can be reduced, but at the cost of more blurry output images [150, 226, 282]. Alternatively, rendering quality can be enhanced by adding back in high frequency components [226]. Sloan *et al.* [217] discussed different speed-up strategies for rendering of light-fields. The main conclusion was that there is always a trade-off between speed and quality, therefore confirming the necessity for efficient post-processes. Liu *et al.* [156] estimate scene geometry dynamically using a color similarity-based plane sweeping algorithm. Another possibility is to make use of dynamic textures [48]. Here a coarse geometry is used to capture large scale variations in the scene, while the residual statistical variability in texture space is captured using a PCA basis of spatial filters. It can be shown that this is equivalent to the analytical basis. New poses and views can then be reconstructed by first synthesizing the texture by modulating the texture basis and then warping it back onto the geometry. However for a good estimation of the basis many input images are needed.

Nevertheless, these approaches still introduce at least some image blur or visible artifacts if the scene is undersampled and good results can only be obtained by using many images, far more than the settings we are aiming for with our approaches. Therefore the challenge is generating a perceptually plausible rendering with only a sparse setup of cameras.

### GEOMETRY AIDED IBR

Instead of relying on sampling density, another approach to increase rendering quality from sparsely sampled image data is to make use of a geometry proxy representing the scene. In Lumigraph Rendering by Gortler *et al.* [99] the 4D Light Field function is depth corrected by such a proxy. Adopting a new

weighting function for the input cameras the Lumigraph can be extended to unordered sets of cameras [36]. For object centered input views a similar approach was previously published by Pulli *et al.* [191] combining proximity to the input views, deviation from the normal and distance to mesh boundaries in the weighting scheme, plus Pulli *et al.* use a depth map per image instead of a single geometry proxy. In Debevec *et al.*'s View-dependent Texture Mapping [59] the geometry proxy is created by hand and later refined by a stereo approach in an offline rendering tool. Removing the model-based stereo part, real-time rendering became possible [58] and incorporating different layers per view allows for more artistic control [196]. Isaksen *et al.* [121] show that if scene depth estimation is precise enough and no occlusion occurs, any single scene element can, in theory, be reconstructed without ghosting artifacts by applying a *wide-aperture filter* combining the influence of more than the classical four input cameras. To reconstruct human actors Carranza *et al.* [40] made use of a parameterized human model that was optimized to fit the given silhouette constraints from each camera. The choice, which input camera should be used for texturing, is based on the normal of the template model, therefore no view-dependent effects can be captured and artifacts might appear if the used input cameras change abruptly.

An interesting approach which not only blends colors on a geometry proxy but attempts to reconstruct a consistent, view-dependent geometry of the scene based on billboards and with the aid of bilateral filtering was presented by Waschbüsch *et al.* [258]. German *et al.* [91] propose a very specialized case for sports events. To estimate the players on the field, background subtraction can be applied. Each articulated part of the player can be represented by a single billboard, as the bone structure is known. While it is a crude approximation using articulated billboards, it is sufficient in this case as the players are quite small in the output view. Hornung *et al.* [117] reconstruct depth maps per view using larger particles that adopt to the scene geometry to allow for silhouette aware sampling. Tung *et al.* [246] combine 3D reconstruction with super-resolution to achieve high-quality meshes and textures. By discarding any color inconsistent samples from inter- as well as intra-video frames and reconstructing the images using a Markov random field formulation they are able to produce very clean results free of noise or ghosting. But on the other hand any view-dependent effects are removed as well and computation time increases.

To deal with small calibration errors (around 1.6 pixels or 10mm) Starck *et al.* [222] make use of silhouette as well as stereo data to reconstruct an optimized surface. For the stereo correspondences, they define a larger search range around the epipo-

lar lines to compensate for the calibration errors. However the rendering is done by classic view-dependent texture mapping and therefore reveals artifacts again. All of these approaches require non negligible preprocessing times of several seconds per output frame up to several minutes and are therefore not applicable to real-time applications.

#### IMAGE-BASED MODELING

Image-based modeling extends the notion of image-based rendering in that high-quality 3D geometry scans of an object are augmented with a collection of photos to capture the visual appearance [22, 142, 199, 270].

Acquiring these detailed models is time-consuming and, of course, is possible only for scenes that hold still during acquisition. In any case, image calibration inaccuracies, subcritical sampling, and geometry acquisition errors remain potential sources of rendering artifacts.

#### OCCLUSION HANDLING

In settings with very sparse camera setups, occlusion and registration errors result in disturbing rendering artifacts. Carranza *et al.* [40] therefore proposed to use a visibility map, computing visibility for every vertex of the mesh from several camera views that are slightly displaced in the image plane. Lensch *et al.* [142] search for depth discontinuities to discard samples close to them, as they are prone to errors. Both assume well color calibrated input images, as otherwise sudden changes in the choice of input cameras from one pixel to the neighboring pixel would become annoyingly visible.

In the video view interpolation by Zitnick *et al.* [281] these discontinuities are rendered separately using Bayesian image matting to compute fore- and background colors along with opacities. Extensive pre-processing is needed to achieve interactive rendering frame rates. Matusik *et al.* [168] use opacity hulls to capture also semi-transparency and complex silhouettes. Their approach shows how difficult a high-quality acquisition of even static models can be. Such setups are difficult to built, calibrate and are not always affordable.

#### WARPING TECHNIQUES

Beier *et al.* [23] proposed to use a line-based warping method to interpolate between two images, known from its use in Michael Jackson's music video "Black & White". While the results are obviously very good, all correspondences were acquired by manual adjustment, which is not only very tedious but prohibitive to use in any interactive free-viewpoint renderer. A physically-valid view synthesis by image interpolation is proposed by Seitz



*et al.* [207, 208]. They determine the fundamental matrix to estimate dense-disparity and interpolate between two views of a static scene. Manning and Dyer [163] extend their approach by segmenting different motion layers by hand and restricting motions to rigid-body translations. A similar approach is presented by Xiao *et al.* [273], also based on manual segmentation but that is also able to address non rigid-deformations.

If complete correspondences between image pixels can be established, accurate image warping becomes possible [44]. Mark *et al.* [164] followed the seminal approach of Chen *et al.* [44] but also handled occlusion and discontinuities during rendering. Porquet *et al.* [188] use projective texture mapping to reproject geometric detail onto a coarse geometry. While useful to speed up rendering performance, their approaches are only applicable to synthetic scenes. Often, however, current methods for automatic camera calibration and depth acquisition are too imprecise for these approaches to work on real-world data.

For very similar images, optical flow techniques have proven useful [116, 159]. Highly precise approaches exist, which can be computed at real-time or (almost) interactive rates [35, 187, 263]. They can be employed to create smooth morphs between images, as was used, e.g., by Vedula *et al.* [251] for spatio-temporal view interpolation. In the context of creating virtual walkthroughs the work of Aliaga *et al.* [9, 10] created beautiful results, by capturing a very dense set of omnidirectional images, with an average distance between capturing positions of 4 cm. For image synthesis, they identify corresponding features in the different images, triangulate them in one reference view, and warp this mesh according to the current viewpoint and the interpolated feature positions.

For wider baselines Stich *et al.* presented a perception-based image interpolation technique [227, 228, 229]. They establish a dense correspondence field by an optimized edge-matching used for homography estimations of image patches and smoothed by an anisotropic diffusion process. The resulting flow fields can then be used to interpolate between arbitrary video streams in space and time. Lipski *et al.* [154] make use of high resolution images that provide more details to compute dense correspondences based on a combination of sophisticated feature descriptors and Belief Propagation. They use the interpolation technique of Stich *et al.* [228] for the rendering. However high computation times forbid real-time usage for both approaches without excessive preprocessing.

With the Light Field Video Camera Wilburn *et al.* [265, 266] developed a large camera array to capture dynamic light fields. Similar in spirit, Einarsson *et al.* [66] created a complete acquisition system, the so-called light stage 6, for acquiring and re-

lighting human locomotion. Due to the high amount of images acquired both could incorporate optical flow techniques to create virtual camera views in a light field renderer, by direct warping of the input images. As they do not limit the amount of input images simultaneously used during rendering, they need to compute the flows beforehand to make interactive navigation possible.

Aganj *et al.* [3] match feature points in the input images or use already matched feature points from the reconstruction phase and warp their input images accordingly in a preprocess, so they adopt the input images to the imprecise mesh. However this approach does not warp the images according to the viewpoint, hence the result is always limited in the achievable precision. Takai *et al.* [237] deform both, the mesh and the texture coordinates to create a harmonized output image. While the rendering can again be performed in real-time, the optimization itself, is done in a preprocess.

#### IMAGE INTERPOLATION

Correspondence estimation is only one part of an image-based renderer. The image interpolation itself is another critical part. It has been shown in [211], that even if ground truth motion fields are available image blending cannot yield convincing results in the case of occlusion. It is therefore important to either take occlusion separately into account, as we have done in Chapter 11, or to incorporate it directly into the image interpolation algorithm.

Fitzgibbon *et al.* [82] use image-based priors, i.e., they enforce similarity to the input images, to remove any ghosting artifacts. Drawbacks are very long computation times and the input images must be relatively similar in order to achieve good results, so only small disparities can be handled. An acceleration scheme was later proposed by Woodford and Fitzgibbon [271] but was still far from being interactive. Mahajan *et al.* [161] proposed a path-based method for plausible image interpolation that searches for the optimal path for a pixel transitioning from one image to the other in the gradient domain. As each output pixel in the interpolated view is taken from only a single source image, ghosting or blurring artifacts are avoided, but if wrong correspondences are estimated unaesthetic deformations might occur. Linz *et al.* [151] extend the gradient-based approach of Mahajan *et al.* [161] to space-time interpolation with multi-image interpolation based on Graph-cuts and symmetric optical flow.

Goesele *et al.* [95] use ambient point clouds, stochastically distributed points in space along the line of sight for input pixels with uncertain depth, to hide erroneous geometry and parts that could not be sufficiently reconstructed. In the unstructured vi-

deo rendering by Ballan *et al.* [20] the static background of a scene is directly reconstructed, while the actor in the foreground is projected onto a billboard and the view switches at a specific point between the cameras where the transition is least visible. While this works well for moving objects the general scene setup is very limited.

#### TEXTURE RECOVERY

Ahmed *et al.* [7] incorporate BRDF estimation into the free-viewpoint video system [40]. Similar to the Floating Textures approach presented in Chapter 11 they try to optimize the input images by warping. For every point on the surface they estimate the camera for which the viewing ray from the surface point to the camera deviates the least from the normal vector at that position and use this projected color as reference value. They then warp the input image so that it most resembles this view. In addition frame to frame correspondences are computed in a square domain, similar to geometry images [102], to handle object changes over time, e.g. shifting clothes. A similar approach, but in the spherical domain is used by Starck *et al.* [221] to find temporal correspondences for non-model based, non-rigid surfaces of genus-zero-topology. This was later extended to a smooth 3D reconstruction technique with a static texture map [223], in order to save bandwidth in streaming applications.

Tzur *et al.* [247] provide a photogrammetric texture mapping approach from casual images which only approximately resemble the geometry, by estimating local camera parameters for different parts of the object instead of global camera parameters for each input view. This is actually a good example how difficult texturing of imprecise geometry is. Gal *et al.* [88] create a single seamless texture map for approximate objects. Though their approach produces pleasurable texture maps by local warping of the textures, the result might not resemble the original from all possible viewpoints and lighting information might not be handled correctly. As we will show in Chapter 9, static texture recovery algorithms will never be able to precisely reconstruct the appearance of an object if the underlying geometry is not accurate, as this demands features to *flow* on the objects surface, otherwise results will always be only approximately correct.

There are many approaches, which are able to produce high-quality renderings in specific setups and given unlimited pre-processing time, even more techniques for the interested reader can be found in [160, 240]. However, it seems there are very few approaches that can achieve good rendering results in general multiview camera settings and given only approximate geometry with possibly imprecise camera calibration and real-time rendering requirements.



## ERROR ANALYSIS

*A computer lets you make more mistakes faster than any invention in human history - with the possible exceptions of handguns and tequila.*

— Mitch Ratcliffe

## 9.1 INTRODUCTION

Errors or imprecisions in any stage of the image-based reconstruction pipeline negatively influence the later rendering quality. Figure 40 shows some of these errors that are likely to occur during image-based rendering based on multiview projective texture mapping. Imprecise camera calibration and sloppy foreground segmentation result in projecting background onto the foreground object, Figure 40a. Approximate 3D reconstruction can also cause ghosting artifacts, i.e. features appear more than once in the output image, Figure 40b. And if visibility is not taken into account, several pixels in one input image is projected more than once onto the geometry proxy, Figure 40c. In this chapter we look at the causes for these artifacts. Based on this analysis we are able to derive new solution strategies which are presented in Chapters 10 and 11.

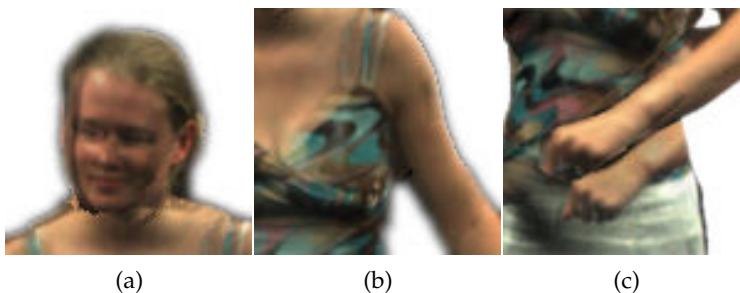


Figure 40.: Different errors in image-based rendering affecting output quality, based on (a) imprecise camera calibration or foreground segmentation, (b) approximate 3D reconstruction, or (c) wrong visibility calculations.

After briefly describing the notations used, we give an overview of the typical error sources degrading image quality in image-based rendering systems, Section 9.2. This is followed by a geometric analysis of the ghosting artifacts, or double images, Section 9.3, including aliasing, Section 9.3.3, ghosting and blurring,

Section 9.3.4 and the band-limiting approach [42, 150] to prevent ghosting artifacts by prefiltering, Section 9.3.5.

### 9.1.1 Notations

We are dealing with a lot of different projections and image domains in this and the following chapters. Hence, it might be beneficial to have a few words on the notation we are using in this part of the thesis. An overview of all symbols used is also given in the appendix A. We denote the name, and also the position of each camera recording the scene with the upper-case boldface letter  $\mathbf{C}$  followed by an index  $i \in \{1, \dots, n\}$ , e.g.  $\mathbf{C}_1$  would be the first input camera,  $\mathbf{C}_2$  the second and so on. The subscript  $v$  denotes the index of the virtual camera  $\mathbf{C}_v$ , which represents the virtual view we want to display to the user.  $\mathbf{I}_i$  denotes the image footage, for the camera with index  $i$ .

A single scene point in the original scene  $\mathbf{G}_O$  is usually denoted as a lower-case boldface character, e.g.  $\mathbf{p}$ . A superscript on  $\mathbf{p}$  denotes the projection into another image domain, e.g.  $\mathbf{p}^v$  would be the pixel position of  $\mathbf{p}$  in the output image  $\mathbf{I}_v$ . To denote a scene point  $\mathbf{p}$  recorded by an input camera  $\mathbf{C}_i$  and back-projected onto the geometry proxy  $\mathbf{G}_A$  we use  $\mathbf{p}_i^{\mathbf{G}_A}$  or simply  $\mathbf{p}_i$  for brevity. Hence, a scene point  $\mathbf{p}$  recorded by a camera  $\mathbf{C}_i$ , backprojected onto the approximate geometry  $\mathbf{G}_A$  and from there projected into another camera  $\mathbf{C}_j$  would be denoted  $\mathbf{p}_i^j$ . The notation  $\mathbf{I}_i^v$  is used to denote the image rendered from a viewpoint  $v$  by projecting the input image  $\mathbf{I}_i$  as texture onto  $\mathbf{G}_A$ .

We will use the same superscript and subscript notation also for other quantities, e.g. the pixelspacing  $\Delta$ .

## 9.2 PROBLEM DESCRIPTION

The plenoptic function  $\mathbf{P}(x, y, z, \theta, \phi)$  describes, in a slightly simplified version, radiance, in other words the color we see, as a function of 3D position in space  $(x, y, z)$  and direction  $(\theta, \phi)$  [2]. The notion of image-based rendering now is to approximate the plenoptic function with a finite number of discrete samples of  $\mathbf{P}$  for various  $(x, y, z, \theta, \phi)$  and to efficiently re-create novel views from this representation by making use of some sort of object geometry proxy.

Any object surface that we choose to display can be described as a function  $\mathbf{G} : (x, y, z, \theta, \phi) \rightarrow (x^w, y^w, z^w)$ , i.e., by a mapping of viewing rays  $(x, y, z, \theta, \phi)$  to 3D coordinates  $\mathbf{p} = (x^w, y^w, z^w)$  on the object's surface. Of course, the function  $\mathbf{G}$  is only defined for rays hitting the object, but this is not crucial since one can simply discard the computation for all other viewing rays. With  $\mathbf{G}_O$  we denote the function of the true surface of the object, and

with  $\mathbf{G}_A$  we denote a function that only approximates this surface, Figure 41, acquired e.g. by one of the methods described in Chapter 2.8.

Next, assuming calibrated cameras, a projection mapping  $\mathbf{P}_i$  exists which describes how any 3D point  $\mathbf{p}$  is mapped to its corresponding 2D-position  $\mathbf{p}^i$  in the  $i$ -th image. From its projected position  $\mathbf{p}^i$  in image  $\mathbf{I}_i$ , the 3D point's color value  $(r, g, b)$  in that image can be read out. Then, in a classic image-based rendering setup [59, 146, 191], the novel view  $\mathbf{I}_{\text{Linear}}^v$  from a virtual camera's viewpoint  $\mathbf{C}_v$  can be synthesized by a weighted linear interpolation scheme

$$\mathbf{I}_{\text{Linear}}^v(x, y, z, \theta, \phi) = \sum_i \omega_i \mathbf{I}_i^v(x, y, z, \theta, \phi) , \quad (9.1)$$

with

$$\mathbf{I}_i^v(x, y, z, \theta, \phi) = \mathbf{I}_i(\mathbf{P}_i(\mathbf{G}_A(x, y, z, \theta, \phi))) \quad (9.2)$$

$$\omega_i = \delta_i(\mathbf{G}_A(x, y, z, \theta, \phi)) \tilde{\omega}_i(x, y, z, \theta, \phi) \quad (9.3)$$

and  $\sum_i \omega_i = 1$  to preserve the overall intensity.  $\delta_i$  is a visibility factor which is one if a point on the approximate surface  $\mathbf{G}_A$  is visible by camera  $\mathbf{C}_i$ , and zero otherwise.  $\tilde{\omega}_i$  is the weighting function which determines the influence of camera  $\mathbf{C}_i$  for every viewing ray, e.g. an angular weighting scheme [59, 191], but without taking visibility into account.

Note that (9.1) is the attempt to represent the plenoptic function as a linear combination of reprojected images. Unfortunately, the plenoptic function  $\mathbf{P}$  is, in general, not representable as a linear combination of the reprojected images. For several reasons simple weighted linear interpolation cannot be relied on to reconstruct the correct values of the plenoptic function in general:

1. Typically,  $\mathbf{G}_O \neq \mathbf{G}_A$  almost everywhere, so the input to (9.2) is already incorrect in most places, Figure 41a.
2. Due to calibration errors,  $\mathbf{P}_i$  is not exact, leading to projection deviations and, subsequently, erroneous color values, Figure 41b.
3. In any case, only visibility calculations based on the original geometry  $\mathbf{G}_O$  can provide correct results. If only approximate geometry is available, visibility errors are bound to occur, Figure 41c.

In summary, in the presence of even small geometry inaccuracies or camera calibration imprecisions, a simple approach based on linear blending is generally not able to correctly interpolate from discrete image samples. While the visibility error results

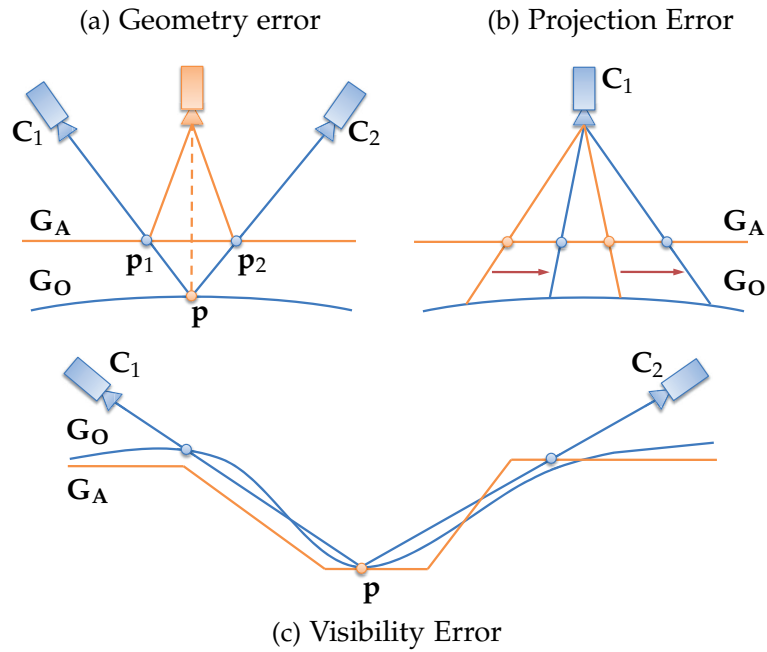


Figure 41.: Error sources: (a) Geometry inaccuracies cause ghosting artifacts. If only the approximate geometry  $G_A$  is available, Point  $p$  on the original surface  $G_O$  is erroneously projected to 3D-position  $p_1$  from camera  $C_1$  and to 3D-position  $p_2$  from camera  $C_2$ . (b) Small imprecisions in camera calibration can lead to false pixel projections (orange lines, compared to correct projections displayed as blue lines). This leads to texture shifts on the object surface and subsequently to ghosting artifacts. (c) Visibility errors. Given only approximate geometry  $G_A$ , point  $p$  is classified as being visible from  $C_1$  and not visible from camera  $C_2$ . Given correct geometry  $G_O$ , it is actually the reverse, resulting in false projections.

in falsely projected color values in the output image, the geometry and calibration errors typically result in so-called ghosting or double images if multiview projective texture mapping is used.

Looking at the geometry error in Figure 41 it also becomes obvious that any approach trying to reconstruct a single texture map from sparse input images and approximate geometry can never create correct results. Because even if one could establish the correspondence between  $p_1$  and  $p_2$ , one would have to choose a static position for  $p$  on  $G_A$ , but if the virtual camera is placed at position  $C_1$  then  $p_1$  would be correct and if it is placed at position  $C_2$  then  $p_2$  would be correct. There simply is *no* correct static texture coordinate position for *any* point on an objects surface if  $G_O$  differs from  $G_A$ .

In the next sections and chapters we will deal separately with all three mentioned error sources and propose methods to handle these. The sources of error for the projection artifacts, based on imprecise camera calibration are quite obvious. The same is



true for the visibility artifacts, based on false visibility tests, due to only approximate reconstruction of the scene. For the derivation of new rendering techniques it is helpful to investigate the geometry error in more detail. In the next section we begin with a geometric analysis of the geometry error. We will then go on and derive techniques for improved, artifact-reduced renderings in Chapter 10 and 11.

### 9.3 A GEOMETRIC ANALYSIS OF GHOSTING ARTIFACTS

In this section we will analyze the causes of ghosting artifacts, i.e., double-images resulting from inaccurate geometry reconstruction. Several authors dealt with the problem of finding the minimum sampling rate for light field rendering, i.e., how many input images are needed to provide artifact-free rendering for a given scene [42, 150]. This problem is important for two reasons. First, the memory requirements for image-based rendering techniques can easily become a bottleneck. Obviously, the less images we capture the less storage and acquisition time is needed. And second, if we know the amount of ghosting in our scene, given our input images and camera parameters, we can find ways to prevent these visual artifacts. Unfortunately, the analyses in [42, 150] dealt with a very restricted set of image-based rendering, namely two-plane parameterized light fields, i.e., all cameras are in equal distance to each other and are facing approximately the same direction. We will try to loosen some of these constraints and derive an analysis suitable for general sparse multiview setups.

#### 9.3.1 Assumptions

To simplify the analysis in the beginning we will make several common assumptions in this section:

- Scene: Occlusion-free and all materials are Lambertian;
- Reconstructed geometry: a simple plane;
- Camera: Camera of limited resolution;
- Interpolation method: Linear interpolation of nearby samples;

As we assume no occlusion in the scene, for now, we can base our analysis on the reconstruction of a single point in the scene. If every point can be correctly rendered, so can the scene. For simplicity of the analysis we will assume a simple plane to represent the approximately reconstructed geometry. The surface

of the scene is assumed to be Lambertian, i.e., light falling on it is scattered in all directions so that the apparent brightness does not change with the viewpoint.

Using our camera model introduced in Section 2.5, we can think of every pixel value as the weighted integral of the light arriving at the image plane of the camera, or in other words, every pixel is a sample of the convolved plenoptic function at the camera center over the viewing angle of the pixel area on the image plane. The support of this filter is simply the angular resolution of the camera.

### 9.3.2 Scene Projection

Let us first take a look at how a single scene point  $\mathbf{p}$  captured by the input camera  $\mathbf{C}_i$  is reprojected onto the image plane of a virtual camera  $\mathbf{C}_v$ . For readability, the illustrations will showcase a two dimensional scene and one dimensional images, Figure 42. In Figure 42a the scene setup is shown. Camera  $\mathbf{C}_i$  records a scene point  $\mathbf{p}$  on the original surface  $\mathbf{G}_O$ . The virtual camera  $\mathbf{C}_v$  records the backprojected contribution of  $\mathbf{p}$  around  $\mathbf{p}_i$  on the approximate surface  $\mathbf{G}_A$ . The  $x$ -axis in Figure 42b-d represents the pixel position in the respective images, 42b and 42c for the input camera  $\mathbf{C}_i$ , 42d for the virtual view. The midpoint of each pixel is shown by the blue dotted lines. The  $y$ -axis is used to depict the amount of contribution of the exemplar scene point  $\mathbf{p}$  in the respective image. As these figures are used for explanatory purposes no concrete values are given.

Assuming unlimited resolution of the input camera, the contribution of a single scene point  $\mathbf{p}$  to the recorded image function of  $\mathbf{C}_i$  is a  $\delta$ -peak. Due to limited resolution,  $\mathbf{p}$  will assign its contribution to the closest pixel in the image  $\mathbf{I}_i$  corresponding to the closest viewing ray of  $\mathbf{C}_i$ , dotted orange line in Figure 42a and 42b. During backprojection the shape of this shifted  $\delta$ -peak changes to a wedge like structure of width  $2\Delta_i$  due to the linear interpolation applied, Figure 42c.  $\Delta_i$  is the distance between neighboring pixel positions on the image plane of  $\mathbf{I}_i$  with normalized focal length for all cameras. It becomes a perspective distorted wedge of width  $w_i$  when reprojected onto the approximate surface  $\mathbf{G}_A$  at position  $\mathbf{p}_i$  and into the virtual camera view at position  $\mathbf{p}_i^v$ , Figure 42d. We call this width  $w_i$  the support of a scene point in the output image. The distortion will increase with increased inclination of the surface with respect to the camera ray. For the further analysis we will neglect this distortion and assume a constant distance between projected pixels on the surface, which would equal an orthographic projection of the cameras. While being a crude approximation for the complete image, we assume the imposed error to be negligible if only

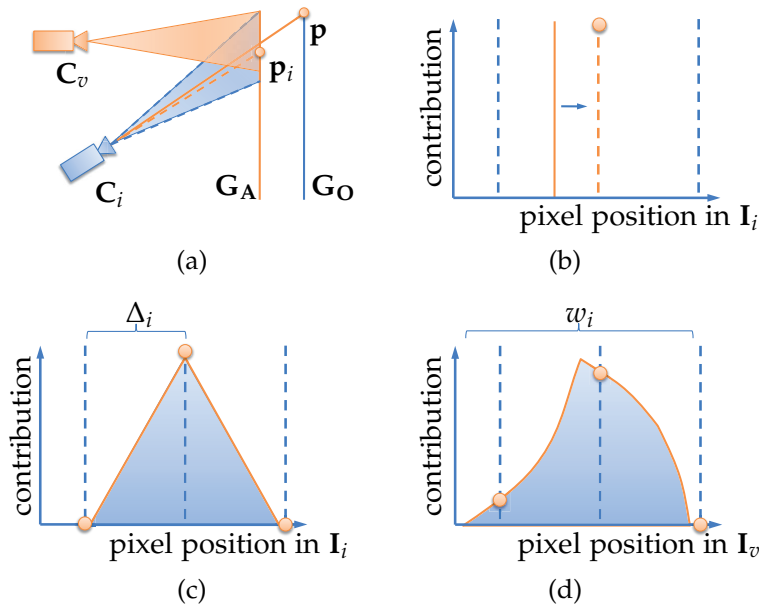


Figure 42.: Scene Projection: (a) Scene setup.  $\mathbf{p}$  is recorded by camera  $C_i$ , backprojected onto the geometry at  $\mathbf{p}_i$  and recorded by camera  $C_v$ . (b) Scene point  $\mathbf{p}$  will contribute to exactly one pixel of the image taken by camera  $C_i$  corresponding to the viewing ray with the smallest angular deviation. (c) During reprojection the influence of  $\mathbf{p}$  in  $I_i$  will change to a triangular shape due to limited camera resolution and linear interpolation. (d) Depending on the inclination angle the wedge becomes projectively distorted in the view of another camera  $C_v$ .

small parts of the image are considered and assuming relatively small inclination angles.

### 9.3.3 Aliasing

The first source of error we are looking at is aliasing. At this stage we assume that no prefiltering of the images was applied after recording. Aliasing in the output image can appear as soon as the projected pixel spacing of the input camera becomes smaller than the pixel spacing of the virtual camera in the virtual camera view [213]. This is also known as undersampling, if seen from the output view perspective. Let  $\Delta_i$  be the pixel spacing in one of the input images  $I_i$  and  $\Delta_v$  be the pixel spacing in the virtual view  $I_v$ , we can calculate the spacing  $\Delta_i^v$  for  $I_i$  in  $I_v$  by taking into account the angle  $\beta$  between the surface normal and the camera, Figure 43. Let  $\beta_i$  and  $\beta_v$  be the angle between the surface normal and the respective cameras. First we calculate the sample spacing  $\Delta_i^{G_A}$  of  $I_i$  on the objects approximate surface:

$$\Delta_i^{G_A} = \frac{\Delta_i}{\cos(\beta_i)} \tag{9.4}$$

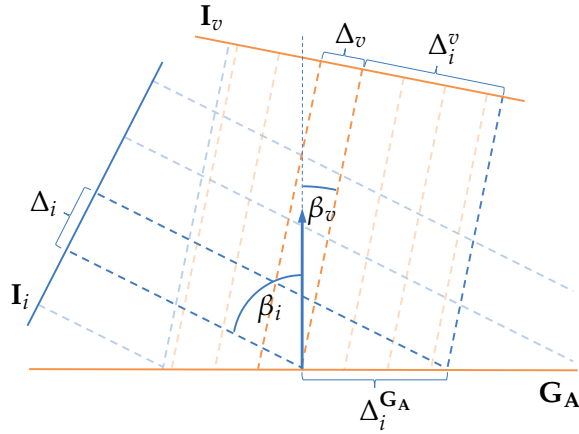


Figure 43.: The relation between the pixel spacings  $\Delta_i$ ,  $\Delta_v$  and  $\Delta_i^v$  in the different views is defined by the angles  $\beta_i$  and  $\beta_v$  between the cameras and the surface, as well as the resolution of the cameras. Here, for orthographic projections.

Hence, the perceived sample spacing  $\Delta_i^v$  of  $\Delta_i^{G_A}$  in the image plane of the virtual camera is

$$\Delta_i^v = \Delta_i^{G_A} \cos(\beta_v) \quad (9.5)$$

Substituting Equation (9.4) into (9.5) we get

$$\Delta_i^v = \frac{\Delta_i}{\cos(\beta_i)} \cos(\beta_v) \quad (9.6)$$

As we do not know anything about the frequencies contained in the input images, we must assume that new aliasing artifacts are introduced as soon as

$$\Delta_i^v < \Delta_v \quad (9.7)$$

Due to the linear interpolation during reprojection the support  $w_i$  of a reprojected scene point equals twice the sample spacing  $\Delta_i^v$ . Therefore, we exhibit aliasing if

$$w_i < 2\Delta_v, \quad (9.8)$$

see Figure 44a. Interestingly classic image-based rendering such as light field rendering [36, 146], seldomly pays attention to this property, as they assume every output pixel to be represented as a single viewing ray instead of a viewing cone defined by the pixel extends. Most prefiltering techniques are based on linear interpolation, as e.g. quadrilinear interpolation in [146], or on band-limiting the input images [42, 150, 226], but this is only a valid interpolation if Equation (9.7) is satisfied. But one can easily create counterexamples. If the camera is moved further away from the scene, this will inevitably decrease  $\Delta_i^v$  for a perspective view. While seldomly mentioned in the image-based rendering literature, this problem is known for decades in classic rendering

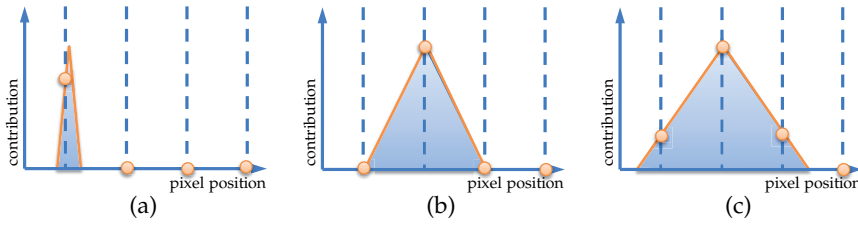


Figure 44.: Spatial Support: (a) If the spatial support  $w$  of a projected scene point in the output image (blue triangle) is too small, the image might exhibit aliasing artifacts. (b) Optimal spatial support. (c) If the spatial support is too large, the image may appear blurry.

and texture mapping, known as minification. And a typical approach to conveniently, though approximately, solve it, is to use mipmaps [268]. Mipmaps can best be thought of as an image pyramid, where for each succeeding level the width and height are reduced by a factor of 2 and each pixel in these reduced levels saves an appropriately filtered version of the original image. During rendering the best matching levels are chosen for the texture look-ups so that the ratio between pixel and texel size is approximately one-to-one. Additionally trilinear filtering can smoothen the transition between different levels.

On the other hand, the image will start to appear blurry as soon as

$$w_i > 2\Delta_v \quad , \quad (9.9)$$

for any input camera  $C_i$ , as the perceived sampling distance of the input camera is lower than the sampling distance of the output view, Figure 44c.

### 9.3.4 Ghosting and Blurring

Ghosting appears whenever a recorded scene point is projected from two or more cameras to different pixel positions in the output view. But as we are dealing with discrete images there is a certain error tolerance which we will take a look at in this section. Let us assume we are given the input cameras  $C_1$  and  $C_2$  as well as a scene point  $\mathbf{p}$  on the original surface  $G_O$  and their projections  $\mathbf{p}_1$  and  $\mathbf{p}_2$  onto the approximate surface  $G_A$  by the input cameras. We can reproject these two into a virtual view  $I_v$  to detect several critical constellations or sources of error. Important for these classes of error are the projected sample spacing  $\Delta_1^v$  and  $\Delta_2^v$  of  $I_1$  and  $I_2$ , and the position of the pixels center projections  $\mathbf{p}_1^v$  and  $\mathbf{p}_2^v$ , i.e., the positions of maximum contribution intensity in the image plane of the virtual camera  $C_v$ . The projected sample spacing  $\Delta_1^v$  and  $\Delta_2^v$  equal half the width  $w_1, w_2$

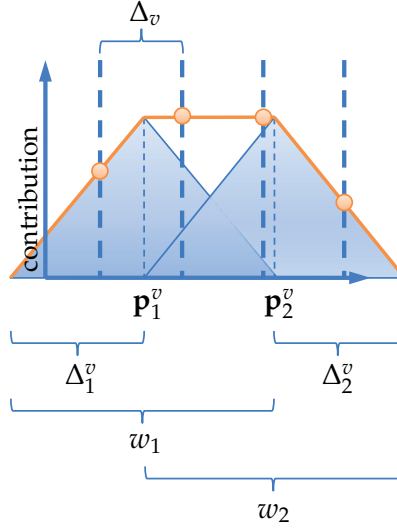


Figure 45.: Scene point  $\mathbf{p}$  is projected into  $I_v$  at position  $\mathbf{p}_1^v$  and  $\mathbf{p}_2^v$  for two cameras  $\mathbf{C}_1$  and  $\mathbf{C}_2$ , respectively. The width  $w_1$ ,  $w_2$  of the projected contribution of a single pixel from  $I_1$  and  $I_2$  equals twice the projected pixel spacing  $\Delta_1^v$  and  $\Delta_2^v$  of  $I_1$  and  $I_2$ .

of the projected contribution of a single pixel, Figure 45. An intuitive definition of ghosting is then:

If the combined contribution of a projected scene point in the image plane of the virtual camera exhibits more than one maximum, ghosting is apparent.

If more than one camera is taken into account that *sees* the scene point  $\mathbf{p}$  and have an influence greater zero, the output image appears sharp, Figure 46a, if

$$\forall i : w_i = 2\Delta_v \text{ and } \forall i, j : \|\mathbf{p}_i^v - \mathbf{p}_j^v\| \leq \Delta_v , \quad (9.10)$$

i.e., the condition for anti-aliased rendering is well satisfied and the projected scene points are no further apart than a single pixel in the output view. The scene point will look blurry, Figure 46b, as soon as

$$\exists i, j : \|\mathbf{p}_i^v - \mathbf{p}_j^v\| > \Delta_v \text{ and } \|\mathbf{p}_i^v - \mathbf{p}_j^v\| \leq \min(w_i, w_j)/2 , \quad (9.11)$$

with  $i \neq j$ . I.e., the projected scene points are further apart than a single pixel but the distance is smaller than half the minimum support. And it will exhibit ghosting, Figure 46c, as soon as

$$\exists i, j : \|\mathbf{p}_i^v - \mathbf{p}_j^v\| > \Delta_v \text{ and } \|\mathbf{p}_i^v - \mathbf{p}_j^v\| > \min(w_i, w_j)/2 , \quad (9.12)$$

with  $i \neq j$ . Hence, artifacts occur if the projected scene points are further apart than a single pixel but the distance is larger than half the minimum support.

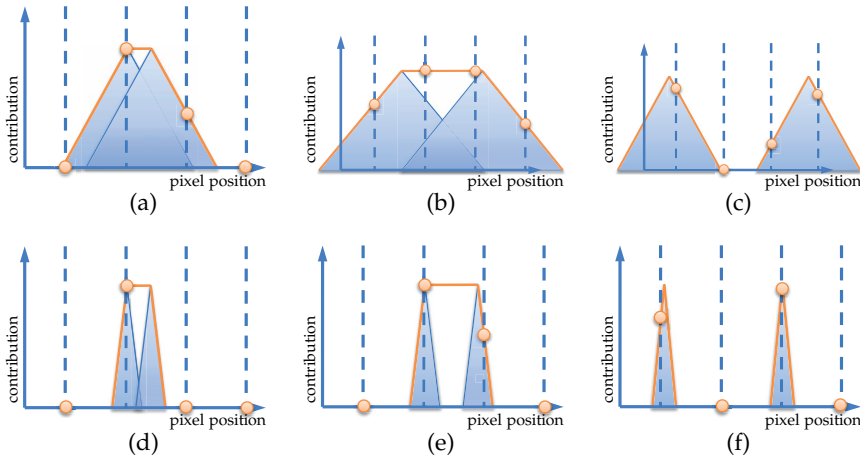


Figure 46.: Rendering quality, here exemplarily shown for two cameras: (a) Almost optimal rendering, no ghosting, no aliasing, the projection is sharp. (b) No aliasing, no ghosting, but the image appears slightly blurry. (c) No aliasing, but ghosting, i.e., the projected scene point is projected to non-connected pixels. (d)-(e) The projected scene point appears sharp, but the image might exhibit aliasing. (f) Worst case, the image exhibits ghosting artifacts plus possible aliasing.

### 9.3.5 Band-limiting

To prevent ghosting and aliasing artifacts in two-plane parameterized light field rendering *band-limiting* was introduced, first mentioned by Chai *et al.* [42] and Lin *et al.* [149]. Chai *et al.* suggested that a sufficient condition for avoiding artifacts altogether is to limit the disparity  $\|\mathbf{p}_i^q - \mathbf{p}_j^q\|$  of all projected scene elements to  $\pm 1$  pixel. If the maximum and minimum scene depth extends,  $z_{min}$  and  $z_{max}$ , are known, one can place the focal plane at the *optimal depth* at

$$z_c = \left[ \frac{1}{2} \left( \frac{1}{z_{max}} + \frac{1}{z_{min}} \right) \right]^{-1}, \quad (9.13)$$

which minimizes disparity. Given the position of the cameras the maximum disparity can be calculated from the optimal depth  $z_c$  and  $z_{min}$  or  $z_{max}$ . If the disparity of some scene elements is greater than  $\pm 1$  pixel, artifacts can be removed by setting the higher frequency part of the spectrum to zero, or, according to our representation, the applied filter should widen the support of the projected scene point so that the combined contribution reveals only a single maximum. Note that due to the quadrilinear interpolation applied in light field rendering only direct neighbors on the camera plane of the light field need to satisfy this condition, while in a more general setup all cameras contributing to the according output pixel need to satisfy it. If this is the

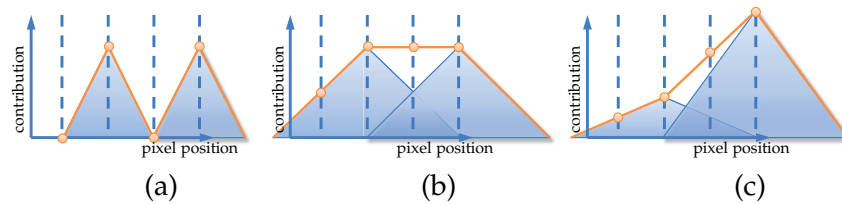


Figure 47.: Band-limited Rendering: (a) Perceived disparity is larger than a single pixel and the support smaller than the required minimum. The resulting image exhibits ghosting artifacts if no further error-handling is applied. (b) By widening the spatial support of the projected scene point via low-pass filtering the input images, ghosting is removed but blur introduced. (c) Changing the weighting of each camera during rendering changes the contribution. No ghosting appears as the combined contribution still reveals only a single maximum.

case, no ghosting occurs even if the weighting of the cameras is changed during rendering, as this basically shifts the peak on the image plane, see Figure 47. Nevertheless, depending on the disparity observed, this approach may remove a lot of details in the input images, resulting in blurred rendering results.

Applying band-limiting to general image-based rendering with multiview projective texture mapping is difficult due to the generalized camera setup and the unknown depth uncertainties occurring from the reconstruction, as well as the unknown frequency limit of the input images. But one can approximate the band-limiting by simply setting the amount of filtering as a user-definable parameter. Comparisons of band-limiting to our approaches presented in this thesis are given in Chapters 10 and 11.



FILTERED BLENDING FOR MULTIVIEW  
PROJECTIVE TEXTURING

---

*The supreme accomplishment is to blur the line  
(between work and play).*

— Arnold Toynbee

## 10.1 INTRODUCTION

Ghosting artifacts are salient and visually disturbing artifacts. Arguably, human perception is much more susceptible to ghosting than to blur [150]. In fact, we are accustomed to seeing blurred objects, e.g. due to motion blur or simple out-of-focus effects. It stands to reason that blur is a more *convenient* artifact than ghosting. Nevertheless blur is an artifact and should be avoided as often as possible. The main drawback of band-limiting discussed in Section 9.3.5 is that the method is based on prefiltering of the input images, and detail is irrevocably lost. However, it seems obvious that if the virtual camera is placed at the same position as one of the input cameras, there is, despite aliasing issues, no need to use the prefiltered version of the input images, as correct results can be obtained with the unfiltered image.

In this chapter we will adapt this idea and extend the former analysis from Chapter 9 and base our anti-ghosting method on the perceived disparity of a single input camera in relation to the virtual camera's position and resolution. This way we are able to incorporate view-dependent filtering. We investigate the band-limiting approach as the upper bound on necessary filtering, but preserve more details in cases where the virtual camera is close to one of the input cameras.

The rest of this chapter is structured as follows: we start with a view-dependent ghosting artifact analysis, Section 10.2. From this we derive our view-dependent filtering approach in Section 10.3. Details of the GPU implementation are given in Section 10.4. We evaluate our approach in Section 10.5, and discuss our results in Section 10.6.

## 10.2 VIEW-DEPENDENT GHOSTING ARTIFACT ANALYSIS

Let us assume we are given a planar scene  $\mathbf{G}_O$ , an approximate scene  $\mathbf{G}_A$  whose normals point in the positive  $z$  direction, plus a geometry offset  $\Delta z$  between the two. Given an input camera  $\mathbf{C}_1$  and a virtual camera  $\mathbf{C}_v$  we can calculate the texture shift in the  $xy$ -plane, Figure 48.

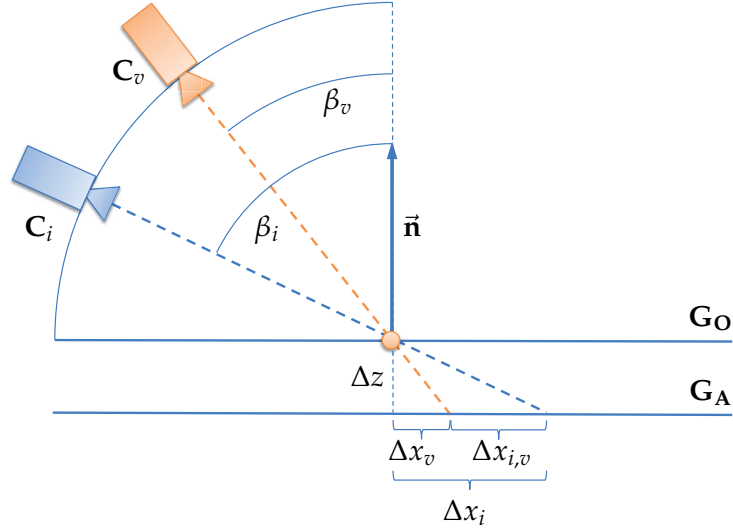


Figure 48.: Absolute and relative shift: The geometry offset  $\Delta z$  results in an absolute texture shift  $\Delta x_i$  on the approximate surface  $\mathbf{G}_A$ . This shift depends on viewing angle  $\beta_i$ . However, the observed, or relative, texture shift  $\Delta x_{i,v}$  does not only depend on  $\beta_i$  but also on  $\beta_v$  and therefore on  $\Delta x_v$  and  $\Delta x_i$ .

Given a viewing direction  $d$  for a camera  $\mathbf{C}_i$ , the shift on the approximate surface  $\mathbf{G}_A$  is given by

$$\Delta \mathbf{x}_i = \begin{pmatrix} \Delta x_i \\ \Delta y_i \end{pmatrix} = \Delta z \begin{pmatrix} \frac{d_x}{d_z} \\ \frac{d_y}{d_z} \end{pmatrix} \quad (10.1)$$

This absolute texture shift is interesting for the analysis of overall quality for one specific camera. If we now add our virtual camera  $\mathbf{C}_v$ , the perceived disparity between these on  $\mathbf{G}_A$  that can be observed in the output view is

$$\Delta \mathbf{x}_{1,v} = \begin{pmatrix} \Delta x_{1,v} \\ \Delta y_{1,v} \end{pmatrix} = \begin{pmatrix} \|\Delta x_1 - \Delta x_v\| \\ \|\Delta y_1 - \Delta y_v\| \end{pmatrix} \quad (10.2)$$

Reprojecting the relative shift  $\Delta \mathbf{x}_{1,v}$  into our virtual view using Equation (9.5) to obtain the perceived relative shift  $\Delta \mathbf{x}_{1,v}^v$  in the output image, Equations (9.5) and (10.2) give us an efficient way of estimating the observed disparity in our output view.

## 10.3 VIEW-DEPENDENT FILTERING

Our goal here is to decouple our previous definition for blurred rendering, Equation (9.11), from the relation between the input cameras in such a way that each input image and the necessary amount of filtering is only dependent on the relation between a single input camera and the virtual camera. Therefore we replace the static components from Equation (9.11) with our view-dependent ones. We substitute the perceived projected distance  $\|\mathbf{p}_i^v - \mathbf{p}_j^v\|$  with our view-dependent relative shift  $\Delta\mathbf{x}_{i,v}^v$  and remove all but one camera from the right-hand side to obtain

$$\|\Delta\mathbf{x}_{i,v}^v\| > \Delta_v/2, \text{ and } \|\Delta\mathbf{x}_{i,v}^v\| \leq w_i/4 \quad (10.3)$$

We need to add the division by 2 on both right-hand sides, as another camera could result in a shift in the opposing direction. This was not needed in (9.11) as we looked at two input views instead of one input view and the virtual camera. If the perceived disparity  $\Delta\mathbf{x}_{i,v}^v$  is smaller than  $\Delta_v/2$ , i.e., half a pixel, and the spatial support is large enough to prevent aliasing, the result should look fine and no further filtering is necessary. A classic example would be if the positions of input camera  $\mathbf{C}_i$  and output view  $\mathbf{C}_v$  almost coincide.

Taking a closer look at Equation (10.3) we see that almost all variables are fixed, due to our necessary specifications.  $\Delta\mathbf{x}_{i,v}^v$  depends on the geometry error  $\Delta z$  which we cannot change, otherwise we would have done so in the 3D reconstruction phase.  $\Delta_v$  is connected to the output resolution and we do not want to change that either. So the only variable left is  $w_i$ . We already know that we can increase  $w_i$  by low-pass filtering of the input image, plus Equation (10.3) gives us a direct estimate of how to filter the image.

To get a better insight on how our view-dependent filtering affects the output image, we can use our notation from Chapter 9 and inspect the contribution of a projected scene point into the virtual view with varying weights for each of the input cameras, Figure 49. If the virtual camera is very close to one of the input cameras almost only the influence of that camera could be used with optimal filtering to prevent aliasing, but no further blurring, Figure 49a. The influence of the second camera approaches zero, while its support is spread out almost unnoticeably in the output image. In the case where the virtual camera is placed directly between two views, Figure 49b, the view-dependent filtering equals the band-limiting approach from Section 9.3.5.

An interesting case occurs if the camera is not placed at any of the two extremes, Figure 49c. In this case both cameras have a noticeable influence on the output view and, according to our definition from Section 9.3 we obtain ghosting, as the combined

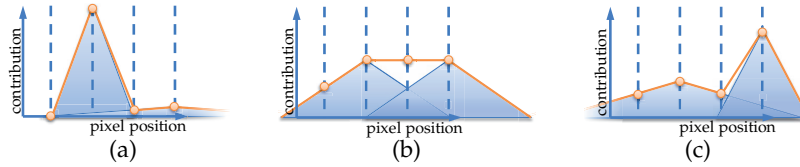


Figure 49.: Influence of view-dependent filtering on the contribution of a single scene point projected by two cameras into the virtual view: (a) The projected and blended input views approach the original input view if the virtual camera coincides with it. (b) If the virtual view is between cameras, filtering resembles the band-limiting approach, cf. Figure 47. (c) For arbitrary camera positions filtering is based on viewing position. The closer the virtual camera is to one of the input views, the less filtering is necessary and the higher the influence of the nearby image.

influence of the projected scene point results in two local maxima! The reason for this is that the decoupling of the input cameras in Equation (10.3) is no direct equivalent to Equation (9.11). Instead of assuring that  $\|\mathbf{p}_i^v - \mathbf{p}_j^v\| \leq \min(w_i, w_j)/2$ , view-dependent filtering asserts  $\|\mathbf{p}_i^v - \mathbf{p}_j^v\| \leq \max(w_i, w_j)/2$ . But otherwise view-dependent filtering would not be possible at all. While this renders any view-dependent filtering-based method theoretically useless, we can still use this approach for improved rendering. The reason is that the texture of the input images is not taken into account in this analysis. Many natural images exhibit a strong correlation between neighboring pixels and this correlation is even increased due to the filtering. Therefore even though the images might theoretically exhibit ghosting, it is not visible in most cases. We will describe the implementation of our view-dependent filtering approach in the following section.

#### 10.4 GPU IMPLEMENTATION

To motivate our implementation, consider the scene setup depicted in Figure 50. The correct scene point  $\mathbf{p}$ , which we would like to render, lies on the line of sight of the viewing ray passing through  $\mathbf{C}_v$  and  $\mathbf{p}_0$ , somewhere within the interval of maximum depth uncertainty  $d_{max}$  from the approximate geometry, given by the intersection points  $\mathbf{p}_1$  and  $\mathbf{p}_2$  with the offset geometry. As in any practical setting we do not know whether the correct geometry offset  $\Delta z$  is in the positive or negative normal direction, we need to deal with both possibilities.  $d_{max}$  must be a user-specified parameter, as most reconstruction techniques do not have an upper or lower bound on the achieved quality. The line segment  $\overline{\mathbf{p}_1\mathbf{p}_2}$  projected into the texture space of camera  $\mathbf{C}_1$  reveals another line segment  $\overline{\mathbf{p}_1^1\mathbf{p}_2^1}$ , which we call the line of dis-

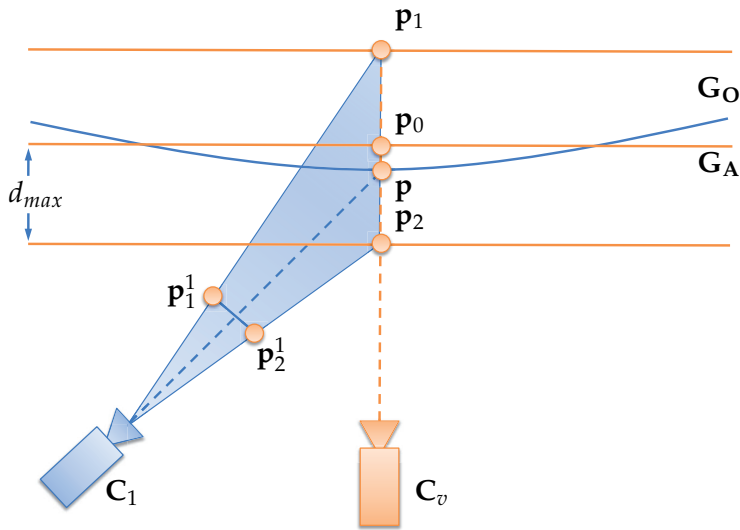


Figure 50.: Scene point estimation: Scene point  $p$  observed from view-point  $C_v$  can only be estimated to lie somewhere between  $p_1$  and  $p_2$ , defined by the maximum depth uncertainty  $d_{max}$ . Its correct color value observed by camera  $C_1$  lies somewhere between the projected texture coordinates  $p_1^1$  and  $p_2^1$ .

parity. Any value on this line could be the correct texture value. This is in fact similar to an epipolar geometry constraint [110].

We solve this uncertainty problem in a resampling process. Choosing  $\frac{p_1^1 + p_2^1}{2}$  as the sampling position and  $\|p_1^1 - p_2^1\| + \epsilon$  as the simulated new sampling distance  $\Delta_1$ , with  $\epsilon \rightarrow +0$ , we anisotropically resample the texture function of  $I_1$  along the line of disparity  $\overline{p_1^1 p_2^1}$ . This way we avoid most ghosting, since the correct texture values always contribute to the corresponding output pixels. As this approach takes the current viewpoint into account, the closer the virtual camera is to one of the input cameras, the fewer frequencies are cut off from that input image and the output image will contain much more details than in a band-limiting approach. If the input camera and virtual camera coincide, all detail is preserved. This way, we implicitly take the input camera distribution into account, as the size of our filter is based on the geometric uncertainty and position of the input cameras.

The depth uncertainty itself can be established in different ways. In two-plane parameterized light field rendering, it is usually the difference along the  $z$ -axis from the focal plane, which is orthogonal to this axis by definition. For synthetic light fields the value of uncertainty might be known in advance, but is to be estimated for real world scenes. Then  $p_1$  and  $p_2$  can be calculated by intersecting every viewing ray with the plane at  $z_{min}$  and  $z_{max}$ , which are the minimum and maximum  $z$ -values in the scene, respectively.

In a more general image-based rendering setup, one could decide to either create an offset along the normal of the approximate surface with which the viewing ray is intersected, or the offset is created along the viewing ray. In the first case the calculated disparity becomes very large at objects silhouettes, as the normal is almost perpendicular to the viewing ray's direction. This leads to strong and distracting blurring artifacts. In addition the filter-size might abruptly change at triangle boundaries if the normal changes, which would again lead to unpleasant visual disturbances. We therefore chose to calculate the offset along the viewing ray. This results in a small blur for images of those cameras close to the current viewpoint and larger blur for those farther away. This works especially well together with an angular metric that computes the influence of each camera for each viewing ray.

Since the support of the applied low-pass filter can theoretically become arbitrarily large, we take two simple steps to alleviate the needed effort. First, we make strong use of GPU processing power. The whole filtering algorithm is implemented as a pair of vertex and fragment shaders. Second, we trade off detail for speed by applying a multi-resolution technique. We set a threshold  $\nu$  for the filter size  $\mu$  in texture space. If this threshold is exceeded we use the  $n^{\text{th}}$  level of the input image-pyramid computed in a preprocess instead of the image itself, with

$$n = \lceil \log_2\left(\frac{\mu}{\nu}\right) \rceil \quad (10.4)$$

In our implementation we set  $\nu = 64$  pixels. Note that this approach has almost no effect on the visual quality of the output, since a large filter size implicates a small weighting factor for an input camera and therefore only a small contribution to the output image, but speeds up the whole rendering process by a factor of roughly three.

Interestingly, depending on the movement of the virtual camera, the constant change of blur in the output image can evoke the impression of repeatedly changing speed, even if a movement is in fact constant. We can solve this perceptual problem by applying a simple motion blur technique using OpenGL's Accumulation Buffer [214]. If the viewpoint does not change, the image quickly converges to the optimally filtered solution.

Our presented algorithm in this section is independent of the weighting scheme used for the image synthesis step, where the projected texture values are combined to reveal the final pixel value. It can be used in conjunction with quadrilinear interpolation [146], the unstructured Lumigraph weighting scheme [36] or angular distance measures [59, 191].

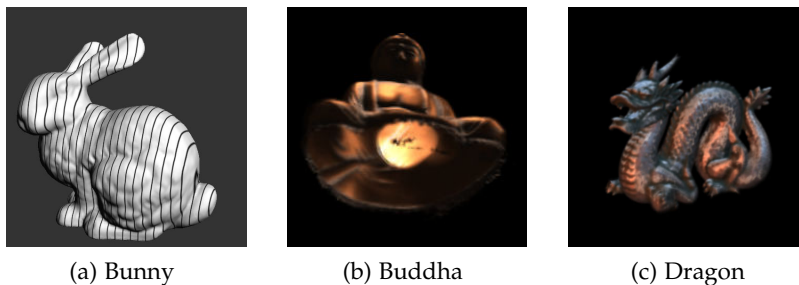


Figure 51.: Images from our test data sets: (a) For the *Bunny*, an approximate 3D geometry model is used. For the synthetic light fields (b) *Buddha* and (c) *Dragon*, a planar surface suffices as geometry proxy. See Table 3 for more information on our test data sets.

	<b>Bunny</b>	<b>Buddha</b>	<b>Dragon</b>
# geometry primitives	948	1	1
Total number of images	19	256	256
Pixels per image	$512^2$	$256^2$	$256^2$
Uncertainty offset	0.63%	7.07%	8.13%
Band-limit filter support	12 pixels	12 pixels	10 pixels
Viewport	$360^\circ \times 360^\circ$	$90^\circ \times 90^\circ$	$90^\circ \times 90^\circ$
Output resolution (pixels)	$512^2$	$512^2$	$512^2$

Table 3.: Information concerning our test data sets shown in Figure 51. The uncertainty offset along the viewing ray in positive and negative direction is given in relation to the diagonal of the geometries' bounding boxes.

## 10.5 RESULTS

We implemented our algorithm on an Nvidia GeForce 8800GTX graphics card using OpenGL and GLSL. For the Filtered Blending approach we add, respectively subtract, the estimated or a priori known depth uncertainty offset along the viewing rays from the vertices positions. Reprojecting the new positions into the different input images yields the needed texture coordinates.

For the resampling process during Filtered Blending, we implemented the Mitchell-Netravali cubic B-spline filter as a fragment shader program [173]. We compared different filters, e.g., a truncated Gaussian and a box filter, and found that the Mitchell-Netravali filter yields the visually most convincing rendering results.

Our test data sets include one classical 3D object, the *Stanford Bunny*, and the two well-known Stanford light fields *Buddha* and *Dragon*. Figure 51 shows example images. Additional data, like the used depth uncertainty or size of the band-limiting filter, are listed in Table 3. To evaluate rendering quality, we compare

our rendering strategies to direct (quadra-)linear interpolation as well as to pre-processed band-limited filtering. For band-limited filtering, the filter support is set to the smallest possible value to prevent ghosting, which depends on the scene. In projective texture mapping, we always select the three nearest cameras for interpolation based on the angular differences of the optical axes compared to the virtual camera. A simple visibility scheme based on shadow mapping [267] is used for each output pixel to exclude the contribution of cameras for which visibility is not given, based on the approximate geometry.

Our first test scene *Bunny* consists of 19 images rendered from randomly selected viewing directions of the original mesh consisting of 65k triangles. Figure 52 depicts the results obtained by the different rendering approaches using only a geometry proxy consisting of 948 triangles. For better rendering quality assessment, some of the details are enlarged in row 2 and 4. Comparing the two leftmost columns in Figure 52, which correspond to (a) standard linear blending and (b) band-limited filtering, to (c) our result on the right, one can note how the ghosting is smoothed away by Filtered Blending, while discontinuities of the texture are much better preserved. We achieve 342 fps when using our Filtered Blending approach.

We also tested our “ghost-busting” approach for light field rendering using the *Buddha* and *Dragon* data sets. Rendering results are shown in Figure 53. Notice how ghosting is prevented in our approach, Figure 53c, while ghosting artifacts are obvious in standard quadrilinear interpolation 53a. At the same time, much finer detail is preserved than if pre-processed band-limited filtering is used 53b. In conjunction with light field rendering, we achieve around 105 fps with our approach.

More results, including real-world examples, can be found in Figure 61 on page 138 of Chapter 11 when compared to our second approach, Floating Textures.

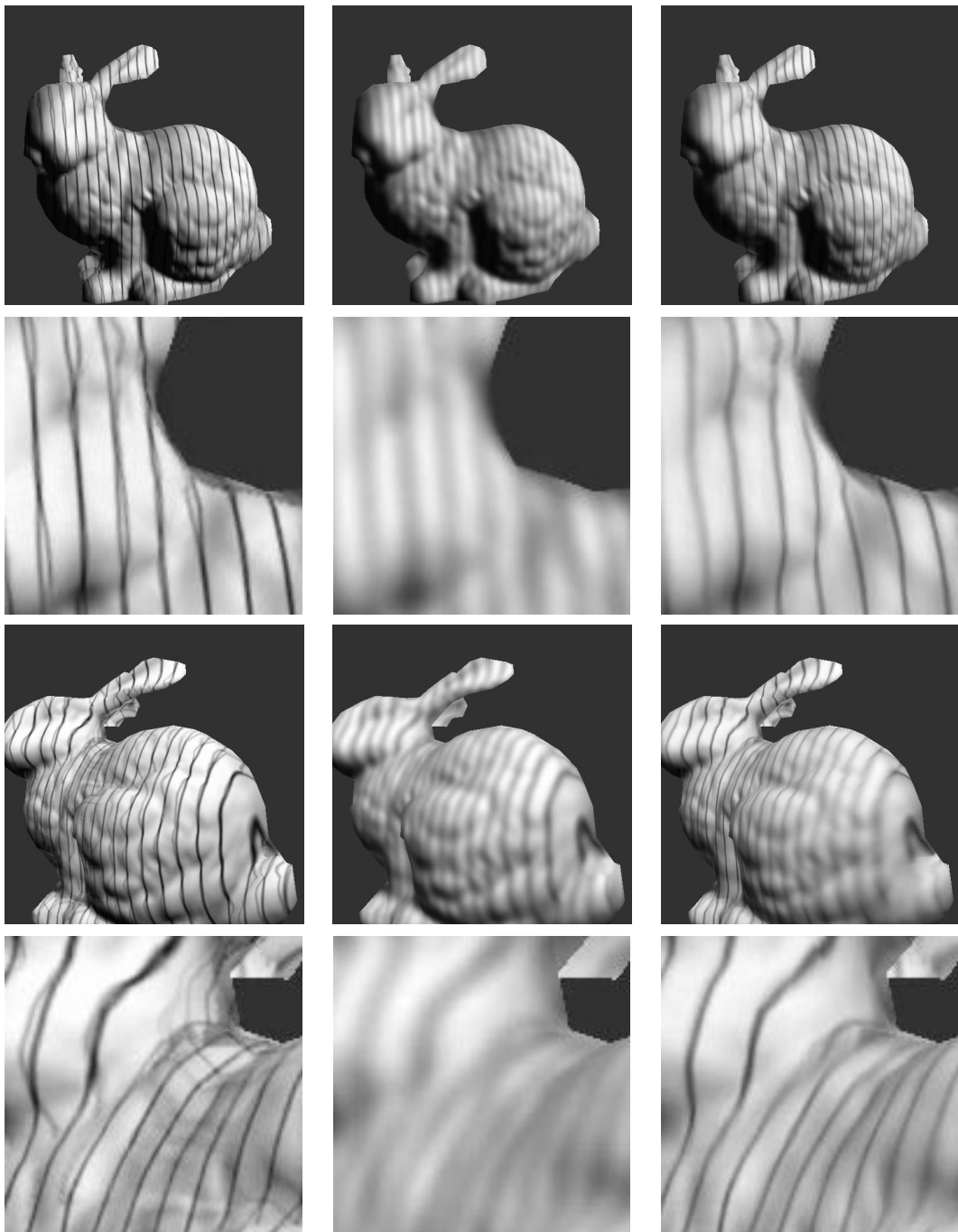
## 10.6 DISCUSSION

In this chapter we have presented an approach to achieve ghosting-reduced rendering results with a viewpoint-optimized low-pass filtering for subcritically sampled light fields, as well as for general projective texture mapping with approximate geometry. In contrast to conventional methods based on prefiltering, our algorithm efficiently diminishes ghosting and better preserves texture details because we are able to take the current viewpoint into account. Real-time rendering performance is achieved on a standard GPU, and the approach can be easily adapted to various different image-based rendering scenarios.



There are, however, certain limitations. If the input samples are too sparse and the geometry reconstruction too imprecise, the image will still look blurry. Small ghosting artifacts might still be visible, as no view-dependent filtering-based approach can remove all artifacts in all cases as explained in Section 10.3. These small artifacts, however, are seldomly visible due to correlation between neighboring pixels in the input images.

In summary, though, Filtered Blending greatly eases the constraints of image-based rendering: coarser 3D geometry and fewer input images are sufficient to still achieve convincing rendering results.



(a) Linear Interpolation      (b) Band-limited filtering      (c) Filtered Blending

Figure 52.: Image-based rendering results for the *Stanford Bunny* with approximate geometry for two virtual camera positions, not coinciding with any input view, row 1 and 3. Row 2 and 4 show close-ups of row 1 and 3, respectively: (a) Linear interpolation reveals strong ghosting around high-frequency details. (b) Band-limited reconstruction removes ghosting, but the result is excessively blurred. (c) Our Filtered Blending approach [71] preserves discontinuities considerably better and reduces artifacts. Notice the much sharper stripes on parts of the bunny.

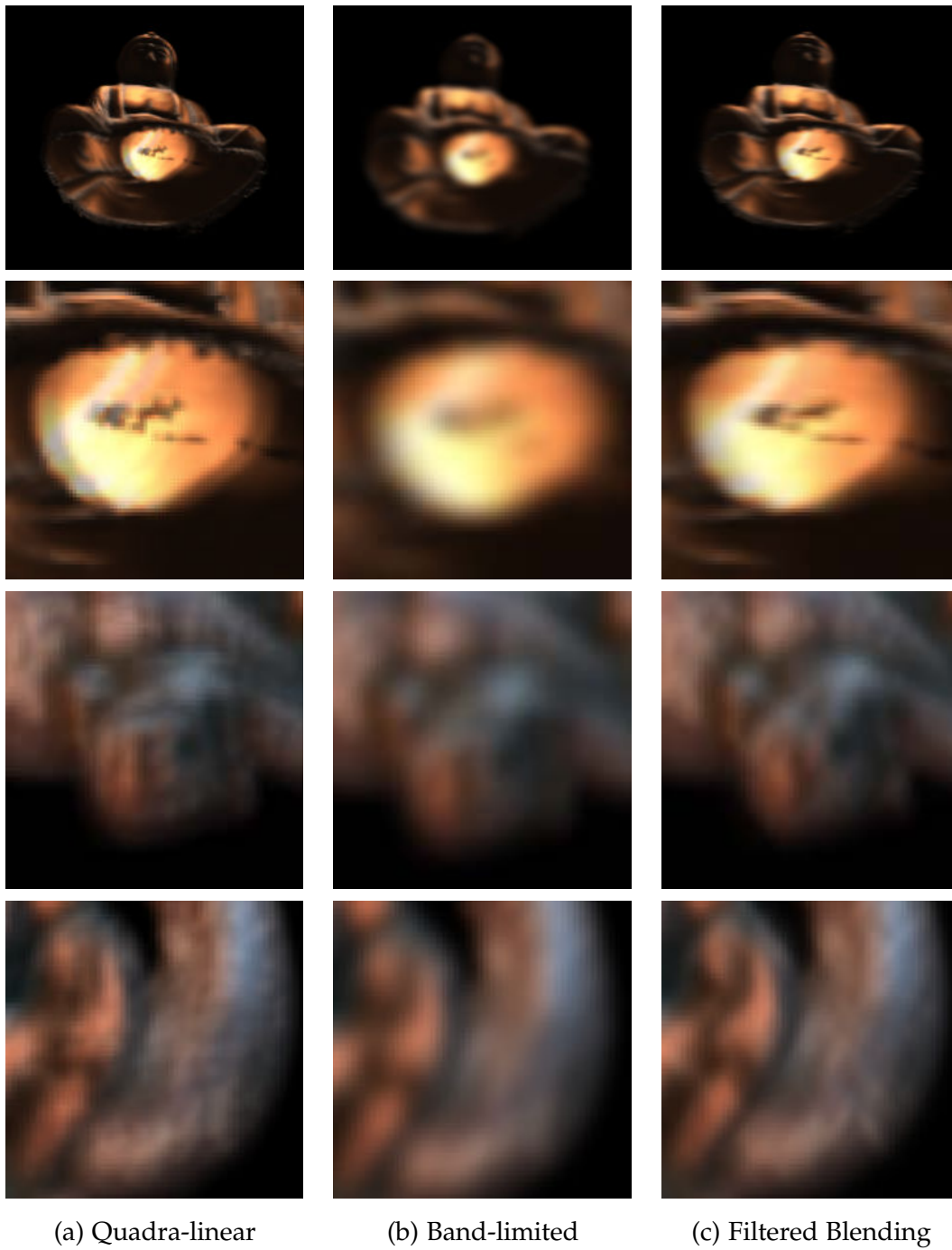


Figure 53.: Comparison for the two sub-critically sampled light fields *Buddha* (top rows) and *Dragon* (bottom rows). Row 2 and 4 show close-up views for better visualization: (a) Quadralinear interpolation cannot suppress ghosting artifacts. (b) Band-limiting the entire light field leads to excessively blurry results. (c) Our Filtered Blending approach preserves more details while ghosting is effectively eliminated.



*Any man is liable to err, only a fool persists in error.*

— Marcus Tullius Cicero

### 11.1 INTRODUCTION

In the last chapter we proposed to remove ghosting artifacts by view-dependent filtering of the input images, as the human visual system is more accustomed to blur than to ghosting artifacts [150]. It turns out that it is even less susceptible to minute shifting [257]. Taking a look at Figure 54, the right image is perceived as more similar to the left image than to the one in the middle.



Figure 54.: The human visual system is more susceptible to local errors than to global ones. Even though the right image, which is a shifted version of the original on the left, has a four times higher sum of squared differences error (SSD) than the blurred one in the middle, we would prefer the shifted version. This is also confirmed by the complex wavelet structural similarity index (CW-SSIM) [257], which gives a better approximation of perceptual image quality. All values have been computed on the luminance channel.

If we make a statistical comparison it turns out that the sum of squared differences (SSD) compared to the left image is four times higher for the right image than for the blurry one in the middle. The blurred image was created by applying a Gaussian blur with standard deviation of  $\sigma = 2.75$  pixels to the original image, while the right image is shifted by ten pixels w.r.t. the original. The image size is  $256 \times 246$  pixels. The reason why we still prefer the shifted one is that the human visual system

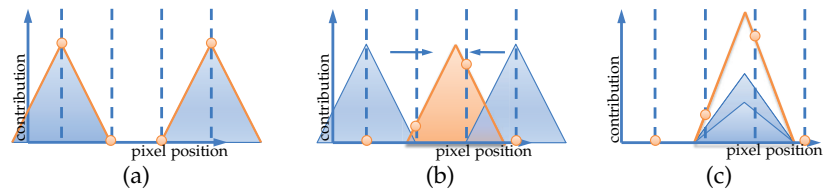


Figure 55.: Floating Textures: (a) The perceived disparity of a projected scene point (blue wedges) is larger than a single pixel resulting in ghosting artifacts. (b) Instead of widening the spatial support, cf. Figure 47 and 49, the idea of our Floating Textures is to shift the position of the projected scene point towards its counterpart and vice versa. The new position (orange wedge) is resulting from a linear interpolation of the positions according to the camera influences. (c) The camera influence changes the height and position of the wedge and therefore the color influence, but all frequencies in the input images are preserved.

is adapted to extract structural information [257], while a global error, like a small shift, passes almost unnoticed.

In addition, filtering-based rendering approaches can only handle ghosting in the occlusion-free case. They are not able to handle visibility errors as described in Chapter 9. Also, camera calibration errors can only be hidden to a certain extend, depending on the amount of blurring of the input images. So removing high-frequencies in the images is not necessarily the best way to go.

In our explanatory test case from the last chapters we projected a single scene point viewed from two cameras into our output view. If one takes a look again at Figure 49, one can see that up to now we have only dealt with the spatial support and the influence of the input cameras, i.e., with the width and height of the projection wedge, respectively. But there is a third component which we have not dealt with yet, the spatial position of the projected scene point. By shifting the projected scene point in the output view, we are able to bring both contributions into congruence, removing ghosting and preserving detail in the images, Figure 55. For extremal views, i.e., if the virtual camera and one of the input cameras coincide, the correct position is known. For all other camera positions, we assume that a linear interpolation of the projected positions provides a sufficient approximation of the real position. By shifting the projected positions we can also handle projection errors by imprecise camera calibration. What we have not dealt with yet are visibility errors, where samples close to occlusion boundaries are falsely projected due to imprecise geometry or calibration errors.

This chapter presents a method to deal with the mentioned artifacts. Because our algorithm runs independently on the graph-

ics card, it can be used in conjunction with many image-based modeling and rendering (IBMR) techniques to improve rendering outcome.

As particular contributions, this chapter presents:

- a novel texturing algorithm that constitutes a symbiosis between classical linear interpolation and optical flow-based warping refinement. It corrects for local texture misalignments and warps the textures accordingly in the rendered image domain;
- a novel weighting and visibility scheme which significantly reduces artifacts at occlusion boundaries;
- a general algorithm that can be applied in conjunction with many IBMR techniques to improve rendering quality;
- an efficient GPU-based implementation of the proposed algorithm which achieves interactive to real-time rendering frame rates;
- a simple extension for static scenes which reduces the actual rendering part to a simple texture look-up.

The remainder of this chapter is organized as follows. In Section 11.2 we describe our Floating Textures as a way to eliminate ghosting and calibration artifacts. Section 11.3 extends this approach to handle also occlusion artifacts. Implementation details are given in Section 11.4, and experimental evaluation results for a variety of different test scenes and IBMR techniques are presented in Section 11.5 before we discuss limitations and conclude with Section 11.6.

## 11.2 FLOATING TEXTURES

In the following, we describe our approach to reduce blurring and ghosting artifacts caused by geometry and calibration inaccuracies using an adaptive, non-linear approach. In a nutshell, the notion of Floating Textures is to correct for local texture misalignments by determining the optical flow between projected textures and warping the textures accordingly in the rendered image domain. Both steps, optical flow estimation and multi-texture warping, can be efficiently implemented on graphics hardware to achieve interactive to real-time performance.

As input, the algorithm requires nothing more but a set of images, the corresponding, possibly imprecise, calibration data, and a geometry proxy. For simplicity, we will first assume an occlusion-free scene and describe how occlusion handling can be added in Section 11.3. Without occlusion, any novel viewpoint can, in theory, be rendered from the input images by warping

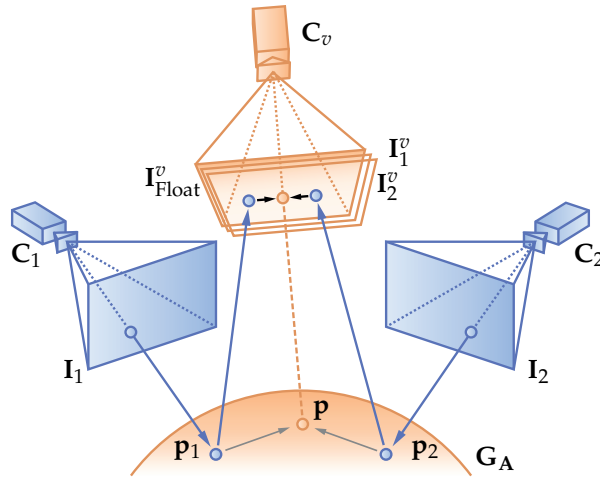


Figure 56.: Rendering with Floating Textures: The input images are projected from camera positions  $C_i$  onto the approximate geometry  $G_A$  and onto the desired image plane of viewpoint  $C_v$ . The resulting intermediate images  $I_i^v$  exhibit mismatch which is compensated by warping all  $I_i^v$  based on the optical flow to obtain the final image  $I_{\text{Float}}^v$ .

[207]. To determine the warp fields, we are safe to assume that corresponding pixels in different images have a set of similar properties, like color or gradient constancy, so that the following property holds:

$$I_j = W_{I_i \rightarrow I_j} \circ I_i \quad , \quad (11.1)$$

where  $W_{I_i \rightarrow I_j} \circ I_i$  warps an image  $I_i$  towards  $I_j$  according to the warp field  $W_{I_i \rightarrow I_j}$ . The problem of determining the warp field  $W_{I_i \rightarrow I_j}$  between two images  $I_i, I_j$  is known as optical flow estimation [116, 159]. For the case when pixel distances between corresponding image features are not too large, algorithms to robustly estimate per-pixel optical flow are available [35].

For our Floating Textures approach, we propose a symbiosis of linear interpolation and optical flow-based warping. We first project the recorded images from cameras  $C_i$  onto the approximate geometry surface  $G_A$  and render the scene from the desired viewpoint  $C_v$ , creating the intermediate images  $I_i^v$ , Figure 56. Note that while corresponding image features do not yet exactly line up in  $I_i^v$  they are much closer together than in the original photos (disparity compensation). This is related to the suggestion of Sawhney [204], who projected the second image of an image pair onto a fronto-parallel plane  $H$  to the first camera to reduce the parallax between the images in order to aid optical flow. We can apply optical flow estimation to the intermediate images  $I_i^v$  to robustly determine the pairwise flow fields  $W_{I_i^v \rightarrow I_j^v}$ .

To compensate for more than two input images, we linearly combine the flow fields according to Equation (11.2) and (11.3),



apply these to all intermediate images  $\mathbf{I}_i^v$  and blend them to obtain the final rendering result  $\mathbf{I}_{\text{Float}}^v$ . To reduce computational cost, instead of establishing for  $n$  input photos  $(n-1)n$  flow fields, it often suffices to consider only the 3 closest input images to the current viewpoint, especially in sparse multiview setups.

We use an angular weighting scheme as proposed in [36, 59] because it has been found to yield better results for coarse geometry than weighting schemes based on normal vectors [40], as stated earlier. Our Floating Textures are, in fact, independent of the weighting scheme used as long as the different weights sum up to 1 for every pixel, which is necessary to ensure that corresponding features coincide in the output image, and given a smooth change of camera influences if the virtual camera moves, otherwise snapping problems could occur.

The processing steps are summarized in the following functions and visualized in Figure 56:

$$\mathbf{I}_{\text{Float}}^v = \sum_{i=1}^n \omega_i (\mathbf{W}_{\mathbf{I}_i^v} \circ \mathbf{I}_i^v) \quad (11.2)$$

$$\mathbf{W}_{\mathbf{I}_i^v} = \sum_{j=1}^n \omega_j \mathbf{W}_{\mathbf{I}_i^v \rightarrow \mathbf{I}_j^v} \quad (11.3)$$

$\mathbf{W}_{\mathbf{I}_i^v}$  is the combined flow field which is used for warping image  $\mathbf{I}_i^v$ . Equation (11.2) is therefore an extension of Equation (9.1) by additionally solving for the non-linear part in  $\mathbf{P}$ .

Note that our Floating Textures deliberately do not satisfy the epipolar constraint anymore. To make use of epipolar geometry constraints one has to presume perfectly calibrated cameras, which is seldom the case. Instead, by not relying on epipolar geometry, Floating Textures can handle imprecise camera calibration as well as approximate geometry, while the minute shifting of the texture on the surface is visually almost unnoticeable.

### 11.2.1 Acceleration for Static Scenes

For static scenes it might seem unnecessary to re-compute the flow fields for every frame. But for a coarse geometry proxy, one cannot simply assign constant texture coordinates to every vertex and every input image. Instead, we propose a slight variation of our Floating Texture generation which can be computed during preprocessing.

Instead of computing flow fields between input images after they have been projected into the image domain of the desired viewpoint, we render the scene from each camera position  $\mathbf{C}_i$  and project all other input images into its image domain, i.e, we render  $\mathbf{I}_j^i$ ,  $j \in \{1, \dots, n\}$ , Figure 57. The flow fields  $\mathbf{W}_{\mathbf{I}_i \rightarrow \mathbf{I}_j^i}$  are then established between the image  $\mathbf{I}_i$  and every  $\mathbf{I}_j^i$ , with  $j \neq i$ .

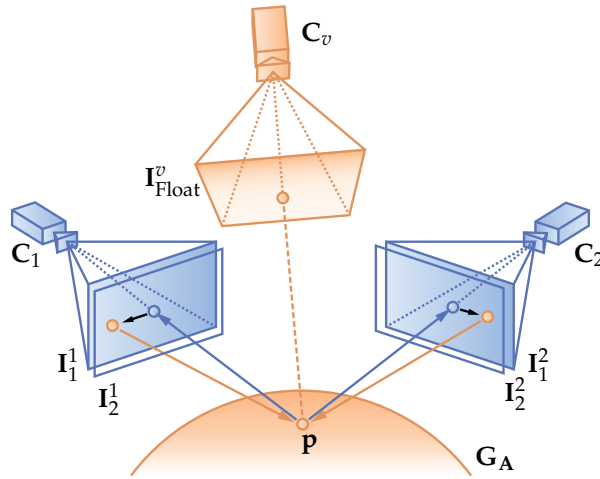


Figure 57.: Rendering with Floating Textures for static scenes: In a preprocessing step, the images  $I_j$  of every camera are projected onto the approximate surface  $G_A$  and into every other input camera  $C_i$ , resulting in the intermediate images  $I_j^i$ . Then the warp fields between these images in every camera view are calculated. For rendering the image  $I_{Float}^v$  from viewpoint  $C_v$ , the warp field of each camera is queried for the texture coordinate offset of every rendered fragment (black arrows), and the corrected texture value is projected back onto the object and into the novel view (blue dot in image plane).

As the views from the cameras do not change over time for static scenes, image synthesis for a new viewpoint reduces to simple projective texturing using warped texture coordinates, Figure 57:

$$I_{Float}^v = \sum_{i=1}^n \left( \left( \sum_{j=1}^n (\omega_j \mathbf{W}_{I_i \rightarrow I_j^i}) \right) \circ I_i \right)^v \omega_i \quad (11.4)$$

Note that in comparison to the viewpoint-centered warping in Equation (11.2) rendering quality may be slightly reduced. On the other hand, the online rendering computations are reduced to two simple texture lookups per fragment and camera.

### 11.3 SOFT VISIBILITY

Up to now, we have assumed only occlusion-free situations, which is seldom the case in real-world scenarios. Simple projection of imprecisely calibrated photos onto an approximate 3D geometry model typically causes unsatisfactory results in the vicinity of occlusion boundaries, Figure 58a: texture information from occluding parts of the mesh project incorrectly onto other geometry parts. With respect to Floating Textures, this not only affects rendering quality but also the reliability of flow field estimation.

A common approach to handle the occlusion problem is to establish a binary visibility map for each camera. This binary

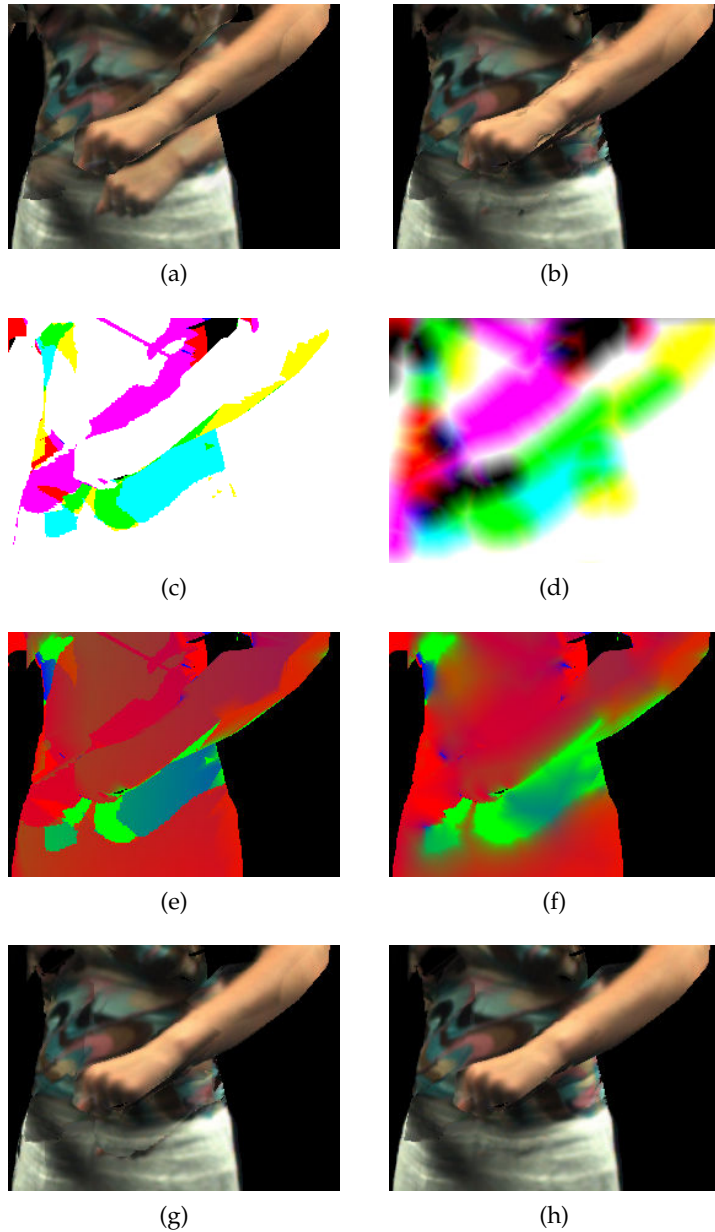


Figure 58.: Visibility artifacts: (a) Artifacts occur if occlusion is ignored. (b) Optical flow estimation goes astray if occluded image regions are not properly filled. (c) Visualization of a binary visibility map from three input cameras. (d) Visualization of a soft visibility map from three input cameras. The amount of filtering is exaggerated for display. (e) Weight map multiplied with the binary visibility map. (f) Weight map multiplied with the soft visibility map, eliminating almost all sudden jumps of camera weights between adjacent pixels. (g) Final result after texture projection using a weight map with binary visibility. (h) Final result after texture projection using a weight map with soft visibility. Note that most visible seams and false projections have been effectively removed.

visibility map is then multiplied with the weight map, which encodes the weight for each camera and pixel. The weights are normalized afterwards so they sum up to one. This efficiently discards occluded pixels in the input cameras for texture generation [40, 142]. But the usage of such binary visibility maps can create occlusion boundary artifacts at pixels where the value of the visibility map suddenly changes, Figure 58 left column. These artifacts are especially noticeable if the cameras are badly color-calibrated.

To counter these effects, we create a *soft* visibility map  $\delta_{\text{soft}}$  for the current viewpoint and every input camera using a distance filter on the binary map:

$$\delta_{\text{soft}}(x, y) = \begin{cases} 0 & \text{if } \delta(x, y) = 0 \\ \frac{\text{occDist}(x, y)}{r} & \text{if } \text{occDist}(x, y) \leq r \\ 1 & \text{else} \end{cases} \quad (11.5)$$

Here  $r$  is a user-defined radius, and  $\text{occDist}(x, y)$  is the distance to the next occluded pixel. If  $\delta_{\text{soft}}$  is multiplied with the weight map, Equation (11.5) makes sure that occluded regions stay occluded, while hard edges in the final weight map are removed. Using this soft visibility map the above mentioned occlusion artifacts effectively disappear, Figure 58h.

To improve optical flow estimation, we fill occluded areas in the projected input images  $I_i^v$  with the corresponding color values from the camera whose weight  $\omega$  for this pixel is highest. Otherwise, the erroneously projected part could seriously influence the result of the Floating Texture output as wrong correspondences could be established, Figure 58b. With hole filling, the quality of the flow calculation is strongly improved, Figure 58h.

#### 11.4 GPU IMPLEMENTATION

The following is a description of our complete algorithm for dynamic scenes. A block diagram is given in Figure 59. The extension to static scenes is straightforward. We assume that camera parameters, input images and a geometry proxy are given. The geometry representation can be of almost arbitrary type, e.g., a triangle mesh, a voxel representation, or a depth map. Even though correct occlusion handling with a single depth map is not always possible due to the 2.5D scene representation.

First, given a novel viewpoint, we query the closest camera positions. For sparse camera arrangements, we typically choose the 3 closest input images. We render the geometry model from the cameras' viewpoints into different depth buffers. These depth maps are then used to establish for each camera a binary visibility map for the current viewpoint, similar in spirit to [40]. We

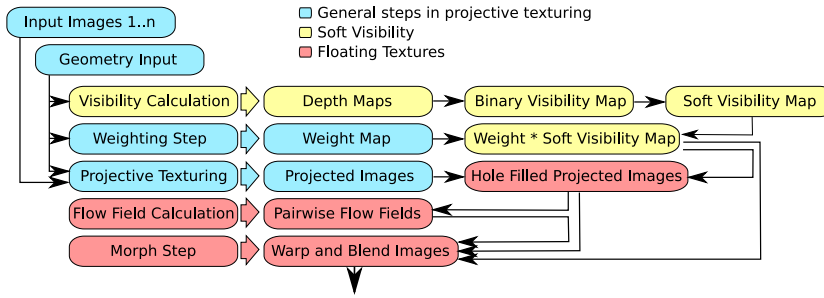


Figure 59.: Complete overview of our Floating Textures algorithm on GPU.

use these visibility maps as input to the soft visibility shader. The calculation of  $\delta_{\text{soft}}$  can be efficiently implemented in a two-pass fragment shader. Next, a weight map is established by calculating the camera weights per output pixel. We use an angular weighting scheme similar to [36]. The final camera weights for each pixel in the output image are obtained by multiplying the weight map with the visibility map and normalizing it so that the weights sum up to one.

To create the input images for the flow field calculation, we render the geometry proxy from the desired viewpoint several times into multiple render targets, in turn projecting each input image onto the geometry. If the weight for a specific camera is zero for a pixel, the color from the input camera with the highest weight at this position is used instead.

To compute the optical flow between two images we rely on our GPU-optimized implementation of the optical flow technique by Brox *et al.* [35]. We found this algorithm to be not only very accurate but also quite robust to noise. Optical flow computation time depends on image resolution as well as on the amount of texture mismatch. Per rendered frame and three input images, we need to compute six flow fields. Even though this processing step is computationally expensive and takes approximately 90% of the rendering time, we still achieve between 5 and 24fps at  $1024 \times 768$ -pixel rendering resolution on an Nvidia GeForce 8800GTX. While we also experimented with faster optical flow algorithms, like a multi-scale implementation of the well known technique by Horn and Schunck [116], we found that results were not as satisfactory. Tests have shown that the limiting speed factor is, in fact, not computational load, but the high number of state changes necessary to compute the flow fields, like shader program or render target switches.

Once all needed computations have been carried out, we can combine the results in a final render pass, which warps and blends the projected images according to the weight map and flow fields.

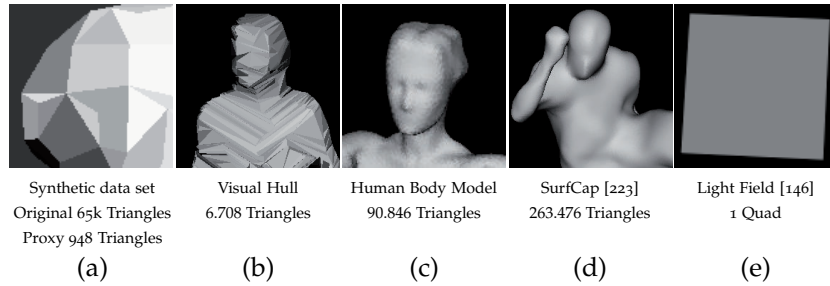


Figure 60.: Geometry proxies corresponding to the different image-based rendering methods evaluated in our experiments, Figure 61.

## 11.5 RESULTS

To evaluate the proposed texturing approach, we have tested Floating Textures in conjunction with a number of different image-based rendering approaches. All tests were carried out using an OpenGL/GLSL implementation of our algorithm on an Nvidia GeForce 8800GTX graphics card. Floating Textures frame rates vary between 5 and 24 fps, depending on the number of input images used and the amount of mismatch between textures which influences the number of iterations needed for the optical flow estimation algorithm to converge. The different image-based rendering approaches with which we evaluated Floating Textures are (Figure 61, from top to bottom):

1. Synthetic Data Set: 49 input images synthesized from a textured 3D model, ground truth available;
2. Polyhedral Visual Hull Rendering: shape-from-silhouette reconstruction [84];
3. Free-Viewpoint Video: a parameter-fitted high-resolution 3D human model [40];
4. SurfCap: high-resolution geometry reconstructed using silhouette, feature, and stereo cues [223];
5. Light Field Rendering: a sub-sampled version of the Stanford Buddha light field data set [146].

Figure 60 depicts the corresponding geometry proxies for each image-based rendering method. For each of these five different image-based rendering techniques, we compared four different texturing approaches (Figure 61, from left to right):

1. Band-limited Rendering [150],
2. Our Filtered Blending [71] from Chapter 10
3. Unstructured Lumigraph Rendering [36], and

#### 4. Our Floating Textures [72], presented in this chapter.

The viewpoint was chosen so that the angular distance to the used input views was maximized.

#### SYNTHETIC DATA SET

The ground truth model of the Stanford Bunny consists of 65k triangles. As input images, we rendered 49 views from random directions applying a colored checkerboard pattern as texture. We then reduced the mesh to 948 triangles to use it as coarse geometry proxy, Figure 60a. Band-limited reconstruction as well as Filtered Blending introduce considerable blurring along texture discontinuities. Unstructured Lumigraph Rendering, on the other hand, leads to ghosting artifacts. Floating Textures, in turn, is able to compensate for most texture mismatch and generates a crisp texture.

#### POLYHEDRAL VISUAL HULL RENDERING

We tested different texturing approaches for the exact polyhedral visual hull reconstruction approach by Franco and Boyer [84], both in an off-line setup as well as in a real-time "live" system.

Our setup of the "live" system consists of 8 cameras with a resolution of 1024x786 pixels that are connected in pairs of two to four PCs (the camera nodes), arranged in an approximate quarter-dome. One PC is used to calculate the approximate geometry of the scene from silhouettes obtained from the camera nodes. The camera nodes calculate these silhouettes by down-sampling the input images and performing background subtraction. The walls of the scene are covered in green cloth to facilitate this process. The approximate geometry is sent to a PC which renders the final image. The rendering algorithm takes the images from 3 cameras to texture the approximate geometry so only these three images are sent over the network to conserve bandwidth. The system allows to capture and render scenes at approximately 10 fps and  $640 \times 480$ -pixel output resolution. Even though the reconstructed visual hulls are only very approximate geometry models, the Floating Textures method is able to remove most of the ghosting artifacts prevalent in Unstructured Lumigraph Rendering, Figure 39 on page 95.

In the offline acquisition setup, we recorded a dancer using eight cameras arranged in a full circle. Excessive blurring is the result if Band-limited Rendering or Filtered Blending is applied, Figure 61. With a linear blending scheme, ghosting and projection errors degrade rendering quality of the face and at the shoulders. These artifacts are efficiently removed by Floating Textures without introducing any additional blur.

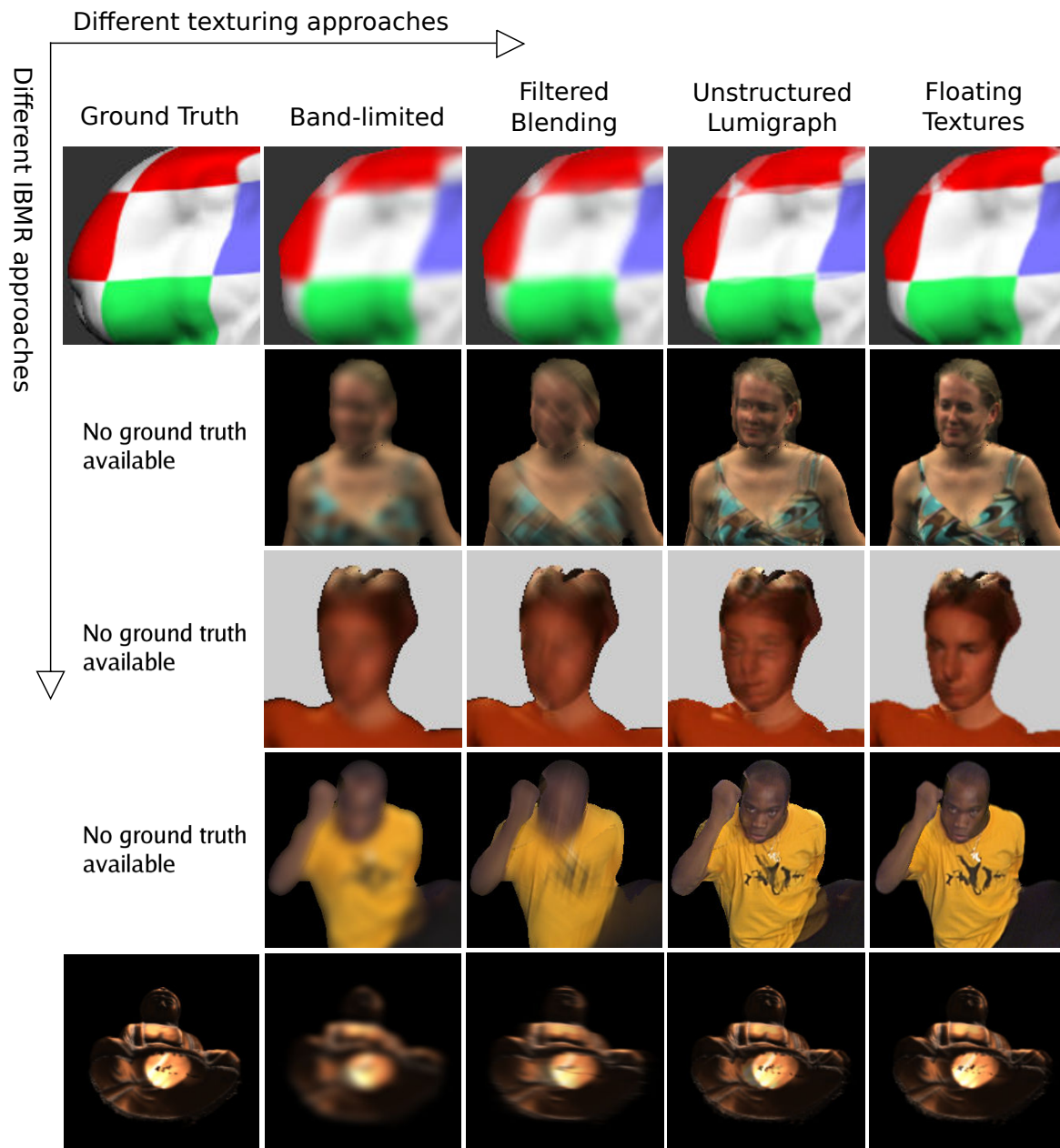


Figure 61.: Comparison of different texturing schemes in conjunction with a number of IBMR approaches. From left to right: Ground truth image (where available), Band-limited Reconstruction [42], our Filtered Blending [71] from Chapter 10, Unstructured Lumigraph Rendering [36], and our Floating Textures [72]. The different IBMR methods are (from top to bottom): Synthetic data set, Polyhedral Visual Hull Rendering [84], Free-Viewpoint Video [40], SurfCap [223], and Light Field Rendering [146].



#### FREE-VIEWPOINT VIDEO

For Free-Viewpoint Video acquisition, eight cameras are regularly spaced around a full circle [40]. Due to the cameras' far spacing, Band-limited Rendering and Filtered Blending eliminate all texture details, Figure 61. Since in Free-Viewpoint Video, a generic 3D model is fit to the video streams by adapting only a good handful of animation parameters, the model surface corresponds only approximately to the person's actual 3D geometry, even though model geometry is very detailed, Figure 60c. This causes noticeable ghosting artifacts if linear blending schemes are applied. The Floating Textures approach corrects for the projective texture mismatch and yields well-defined facial details, Figure 61 right column.

#### SURFCAP

This data set was kindly provided to us from the SurfCap: Surface Motion Capture project [223]. Again, eight cameras are regularly spaced all around a full circle. In computationally elaborate off-line processing, a highly tessellated, smooth geometry mesh is reconstructed, Figure 60d. Far camera spacing prevents Band-limited Rendering and Filtered Blending to preserve details. Even though the mesh consists of  $263k+$  triangles, ghosting and occlusion artifacts still degrade rendering quality if Unstructured Lumigraph Rendering is applied. With Floating Textures, in contrast, virtually artifact-free rendering results are obtained.

#### LIGHT FIELD RENDERING

We down-sampled the original *Buddha* light field data set from  $32 \times 32$  to  $8 \times 8$  images. While Band-limited Rendering indiscriminately blurs away all details, more details are preserved in Filtered Blending, Figure 61. Unstructured Lumigraph Rendering (which corresponds to quadrilinear interpolation for light field rendering) introduces ghosting, as the assumption of dense sampling is violated. The simple planar proxy is not enough to focus the light rays. With Floating Textures, in contrast, we achieve rendering results that are visually almost indistinguishable from the ground-truth. By using the Floating Textures approach in conjunction with light field rendering, comparable rendering results are obtainable from considerably fewer input images.

### 11.6 DISCUSSION

We have presented a new, general method to improve projective texturing using multi-view imagery in conjunction with some 3D geometry proxy. Our Floating Textures approach strongly reduces ghosting and occlusion artifacts and achieves improved

rendering quality from coarse 3D geometry, few input images, and approximate calibration. This saves memory, bandwidth, acquisition time, and money.

While we did not observe any problems during our evaluation experiments, it is obvious that strongly specular surfaces or badly color-calibrated cameras will cause problems for the optical flow estimation. Also, if texture mismatch (ghosting) is too large, e.g. because of very coarse geometry or too few input images, the optical flow algorithm might not be able to find correct correspondences. Robustness of Floating Textures will increase with more sophisticated optical flow techniques. In general, the results with Floating Textures will never be worse than linear interpolation schemes, if the free parameters are adjusted correctly. In our Floating Textures method, we deliberately disregard the epipolar constraint and allow textures to locally *float* in all directions. This way, Floating Textures can compensate for imprecise camera calibration. The small texture shifts on the surface are visually completely imperceptible.

One source for artifacts remaining in rendering dynamic scenes is that of temporally incoherent geometry. In the future, we intend to investigate how Floating Textures might be extended to compensate also for temporal inconsistencies of the geometry proxy. Finally, for highly reflective or transparent objects, motion layer decomposition [205] promises to be another interesting research direction since standard optical flows only generate a single warp field, which cannot represent motion of multiple layers.

Part V.  
Conclusion



*This is a very good question  
and I have a specifically prepared answer for this:  
I don't know.*

— Erik Reinhard

We conclude this thesis with a discussion of our contributions and an elaboration of future research perspectives. We touched on a variety of different topics of computer graphics in this thesis, including seamless image compositing, multiresolution panorama stitching, high-resolution texturing, video matting and multiview projective texture mapping. The unifying idea behind this work was to find ways to conceal some of the common artifacts that occur in the classic pipelines of these algorithms.

In Part II we dealt with the problem of error concealment in the field of seamless image and content synthesis. In Chapter 4 we showed how to create high resolution textures or panoramas from an unordered collection of photographs. We proposed how to find robust correspondences between the images and how to derive a dependency graph depicting the parent-child relation between the images. Dealing with the difficulties encountered when merging different images onto a common image domain, we showed how to deal with color, structure, and resolution differences. From this representation we derived a texture synthesis algorithm to add plausible detail information even to regions not covered by any detail image.

So far we only started to deal with the structural misalignments that can appear if the input images have not been taken from the same viewpoint. This problem requires knowledge about the 3D scene. An inherent problem, however, is that a stronger deformation of the detail images renders the results less plausible in many cases if the deformation is not perfect. This is why we opted for visually less disturbing deformations represented by a homography and a diffusion process. This is sufficient in many cases and moves the possible error to the transition area between the low and high resolution patch, where it can be more effectively hidden.

In Chapter 5 we presented a simple, yet flexible approach to represent multiresolution textures as a hierarchical arrangement of texture patches. In this context we discussed how to

adopt the built-in functionality of current graphics hardware in order to achieve correct filtering and artifact-free rendering results. This way detail insets can be added at arbitrary positions and depth of a texture map, without any change to the underlying 3D model or any z-fighting or flickering, allowing for virtual textures of arbitrary size.

In Part III we proposed a robust matting algorithm for videos. By transforming the problem from the pixel domain to the spectral cluster domain we were able to robustly estimate high-quality mattes for a large number of frames and provided an intuitive and fast to use interface to correct possible errors.

In Part IV we examined common rendering artifacts in free-viewpoint video renderers based on multiview projective texture mapping. In the analysis in Chapter 9 we described how to detect if ghosting is apparent. We used this information to derive our viewpoint-dependent filtering approach in Chapter 10. Extending this approach we finally got to our main contribution in that part of the thesis, the Floating Textures described in Chapter 11. As texture mismatches are the most common cause for ghosting artifacts, we proposed to use a real-time optical flow estimation to match common features on the objects surface and warp the projected images accordingly. In order to deal with artifacts caused by erroneous camera calibration and visibility artifacts caused by the approximate geometry given, we proposed a soft visibility scheme. This scheme weighs the influence of color samples based on their reliability so that the influence of samples in the vicinity of occluding edges is reduced. Combining both techniques, we are able to render plausible in-between views even with a very coarse surface geometry given.

## 12.1 FUTURE WORK

Several ideas for specific directions of future work have already been pointed out in the respective discussions of the previous chapters. By the time this thesis is written, there has already been some further research based on the methods presented in this thesis. Aganj *et al.* [3] proposed an approach which is very similar to the static version of the Floating Textures in Chapter 11. Instead of optical flow they search for robust features to match and interpolate the rest of the warp field by thin-plate splines. Takai *et al.* [237] deform both, the mesh and the texture coordinates to create a harmonized mesh and input textures. Both of them optimize the input images and mesh in a preprocess. In the future we would like to combine Parts II and IV of this thesis into a single approach. The idea here would be to use high-resolution images of a human actor taken beforehand and use them during free-viewpoint rendering to add small de-

tails back into the projected texture. Another direction could be to use a material classification approach, similar in spirit to Ha-Cohen *et al.* [105] and our texture hallucination approach from Chapter 4. Given a large database of different materials, like skin, cloth etc. one could try and segment the images into different material regions and model new details based on the material information provided by the database.

The basic assumption of most techniques in the field of free-viewpoint rendering is still an almost Lambertian scene, as otherwise correspondences are very hard to establish. These problems could be resolved by using more sophisticated illumination and surface models [96], but the requirements with respect to the input data are usually much more stringent, and the computational complexity is prohibitive for real-time applications, at least for the time being.

A hardly investigated research direction is the fusion of algorithmic tasks in multiview video setups with video editing tools. One interesting direction could be a multiview matting algorithm whose results influence the 3D reconstruction which on the other hand affects the correspondence estimation and vice versa. The computational load for such fused algorithms would be extremely high and challenging. Fast and robust methods would be needed, possibly aided by a human-in-the-loop concept. Especially for high-quality productions, an algorithm which works 98% of the time but fails at 2%, is almost useless if no user interaction is provided for to correct errors. Such a unification could lead to a better understanding of how the common problems in multiview video reconstruction and editing are related.





Part VI.  
Appendix



## NOTATION

SYMBOL	DESCRIPTION
$\mathbf{p}$	Point in ND space or general ND vector
$(x, y)$	$x, y$ position in 2D space
$\mathbf{I}$	A digital image
$\mathbf{I}(x, y), \mathbf{I}(\mathbf{p})$	Pixel value of image $\mathbf{I}$ at position $(x, y)$ or $\mathbf{p}$ respectively
$\mathbf{I}_i$	$i$ -th image in a collection of images, e.g. a video, or image corresponding to camera $i$
$\mathbf{I}_i^j$	$i$ -th image in a collection of images, or image corresponding to camera $i$ , warped into the image domain of $\mathbf{I}_j$ using either a homography or a geometry proxy
$\mathbf{I}_i^{j,l}$	Image $\mathbf{I}_i$ warped into the image domain of $\mathbf{I}_j$ at resolution level $l$ . $l = 0$ would be the original resolution of $\mathbf{I}_j$ . $l = 1$ would be the next higher level with twice the width and height and so on
$\mathbf{I}_{i,a}$	Color channel $a$ of image $\mathbf{I}_i$
$\nabla$	Gradient operator
$\nabla_x, \nabla_y$	Gradient operator in $x$ -, $y$ -direction, respectively
$\nabla^2$	Laplacian operator
$\mathbf{H}_{\mathbf{I}_i \rightarrow \mathbf{I}_j}$	A homography warping $\mathbf{I}_i$ into $\mathbf{I}_j$
$\mathbf{N}(x, y)$	Pixel-neighborhood in a synthesized image $\mathbf{S}$ centered at position $(x, y)$
$\mathbf{N}(m(x, y))$	Pixel-neighborhood of an image in a given image collection that resembles most the pixel-neighborhood of image $\mathbf{S}$ centered at position $(x, y)$
$\mathbf{N}(m(x, y)^k)$	The $k^{\text{th}}$ best matching neighborhood of image $\mathbf{S}$ centered at position $(x, y)$ in an image in a given image collection

SYMBOL	DESCRIPTION
$\mathbf{W}$	General warp function
$\mathbf{W}^F$	Forward warp function
$\mathbf{W}^B$	Backward warp function
$\mathbf{W}_{\mathbf{I}_i \rightarrow \mathbf{I}_j}$	Warp function to warp $\mathbf{I}_i$ into $\mathbf{I}_j$
$\circ$	Warping operator
$\mathbf{C}_i$	The position or identifier of the $i$ -th camera in a collection of cameras
$\mathbf{C}_v$	The position or identifier of the virtual camera
$\mathbf{p}^w, \mathbf{p}$	Point in world coordinates, $\mathbf{p}$ is usually used for brevity
$\mathbf{p}_i$	World coordinates of $\mathbf{p}^w$ recorded by $\mathbf{C}_i$ and backprojected onto the geometry proxy
$\mathbf{p}^i$	Image coordinates of point $\mathbf{p}^w$ projected into image $\mathbf{I}_i$ recorded by camera $\mathbf{C}_i$
$\mathbf{p}_i^j$	Image coordinate of point $\mathbf{p}^w$ recorded by camera $\mathbf{C}_i$ and reprojected into image $\mathbf{I}_j$ using a geometry proxy or warping function
$\mathbf{P}(x, y, z, \theta, \phi, t, \lambda)$	Plenoptic function
$\mathbf{P}(x, y, z, \theta, \phi)$	Simplified plenoptic function
$\mathbf{P}$	Abbreviation for the plenoptic function
$\mathbf{P}_i$	Projection matrix according to image $\mathbf{C}_i$
$\mathbf{G}_O$	Original geometric surface
$\mathbf{G}_A$	Geometric proxy or approximate surface
$\Delta_i$	Distance between two neighboring pixels on the image plane of image $\mathbf{I}_i$ assuming a normalized focal length
$\Delta_i^j$	Distance between two neighboring pixels of image $\mathbf{I}_i$ reprojected into the image plane of image $\mathbf{I}_j$ using a geometry proxy or warping function and assuming normalized focal length
$w_i$	Spatial support of a scene point reprojected from image $\mathbf{I}_i$ onto the image plane of the outputview
$\omega_i$	Weighting factor for a specific viewing ray in $\mathbf{I}_i$ including the visibility factor
$\overline{\mathbf{p}_1 \mathbf{p}_2}$	A line segment starting at $\mathbf{p}_1$ and ending at $\mathbf{p}_2$

## PHOTO CREDITS

I would like to thank the following people for providing their photographs under the creative commons licenses or public domain or provided us with helpful material:

NAME	PHOTO / MATERIAL
Vicky Brock (brockvicky)	Two of the Big Ben images in Figure 29
Harshil.Shah	One of the Big Ben images in Figure 29
Bruno Girin	One of the Big Ben images in Figure 29
ricoeurian	One of the Big Ben images in Figure 29
Paul Walker (spratmackrel)	One of the Big Ben images in Figure 29
13bobby	One of the Big Ben images in Figure 29
Adalberto.H.Vega	One of the Big Ben images in Figure 29
Ian Mutto	One of the Big Ben images in Figure 29
ReservasdeCoches.com	Two of the Big Ben images in Figure 29
Lloyd Morgan (fakelvis)	One of the Big Ben images in Figure 29
Dan Lewry (danlewry)	One of the Big Ben images in Figure 29
NR Acampamentos	One of the Big Ben images in Figure 29
antony kelly (apdk)	One of the Big Ben images in Figure 29
Paolo Camera	Two of the Big Ben images in Figure 29
August (cornfed1975)	One of the Big Ben images in Figure 29

NAME	PHOTO
Mark Hillary	One of the Big Ben images in Figure 29
Worawit Suphamungmee (wsuph001)	One of the Big Ben images in Figure 29
jazpillaga	One of the Big Ben images in Figure 29
Wikimedia Commons	Wheat Field with Crows in Figure 28
Y.Y. Chuang	The two test scenes Amira in Figure 37a and Kim in Figure 38.
J. Starck	The SurfCap test scene and model shown in Figure 60 and 61
N. Ahmed	The Free-Viewpoint Video test scene and model shown in Figure 60 and 61
The Stanford University	The Light Field data sets <i>Buddha</i> and <i>Dragon</i> and the 3D model for the <i>Bunny</i> shown in Figure 51, 52, 53, 60 and 61.

## BIBLIOGRAPHY

---

- [1] A. Adams, N. Gelfand, and K. Pulli. Viewfinder alignment. *Computer Graphics Forum*, 27(2):597–606, 2008.
- [2] E. H. Adelson and J. R. Bergen. The Plenoptic Function and the Elements of Early Vision. *Computational Models of Visual Processing*, pages 3–20, 1991.
- [3] E. Aganj, P. Monasse, and R. Keriven. Multi-view texturing of imprecise mesh. In *Asian Conference on Computer Vision*, pages 468–476, 2009.
- [4] A. Agarwala. Efficient gradient-domain compositing using quadtrees. *ACM Transactions on Graphics*, 26(3):1–5, 2007.
- [5] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive Digital Photomontage. *ACM Transactions on Graphics*, 23(3):294–302, 2004.
- [6] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. In *ACM Transactions on Graphics*, pages 584–591, 2004.
- [7] N. Ahmed, C. Theobalt, M. Magnor, and H.-P. Seidel. Spatio-temporal registration techniques for relightable 3D video. In *International Conference on Image Processing*, pages 501–504, 2007.
- [8] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A. K. Peters, Ltd., 3rd edition, 2008. ISBN 987-1-56881-424-7.
- [9] D. G. Aliaga, T. Funkhouser, D. Yanovsky, and I. Carlbom. Sea of images. In *IEEE Visualization*, pages 331–338, 2002.
- [10] D. G. Aliaga, D. Yanovsky, T. Funkhouser, and I. Carlbom. Interactive Image-Based Rendering Using Feature Globalization. In *Symposium on Interactive 3D graphics*, pages 163–170, 2003.
- [11] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.

- [12] C. Ancuti, T. Haber, T. Mertens, and P. Bekaert. Video enhancement using reference photographs. *The Visual Computer: International Journal of Computer Graphics*, 24(7):709–717, 2008.
- [13] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. SCAPE: shape completion and animation of people. *ACM Transactions on Graphics*, 24(3):408–416, 2005.
- [14] N. Apostoloff and A. Fitzgibbon. Bayesian video matting using learnt image priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 407–414, 2004.
- [15] C.J. Armstrong, Brian L. Price, and William A. Barrett. Interactive segmentation of image volumes with live surface. *Computer Graphics*, 31(2):212–229, 2007.
- [16] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.
- [17] X. Bai, J. Wang, D. Simons, and G. Sapiro. Video snapshot: Robust video object cutout using localized classifiers. *ACM Transactions on Graphics*, 28(3):1–11, 2009.
- [18] R. Bajcsy and S. Kovačič. Multiresolution elastic matching. *Computer Vision, Graphics and Image Processing*, 46(1):1–21, 1989.
- [19] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M. Black, and R. Szeliski. A Database and Evaluation Methodology for Optical Flow. In *International Conference on Computer Vision*, pages 1–8, 2007.
- [20] L. Ballan, G. J. Brostow, J. Puwein, and M. Pollefeys. Unstructured video-based rendering: Interactive exploration of casually captured videos. *ACM Transactions on Graphics*, pages 1–11, 2010.
- [21] S. Barret. Sparse virtual textures. Talk at Game Developers Conference, 2008. Available online at <http://www.silverspaceship.com/src/svt/>, visited in Feb. 2011.
- [22] A. Baumberg. Blending images for texturing 3D models. In *British Machine Vision Conference*, pages 404–413, 2002.
- [23] T. Beier and S. Neely. Feature-based Image Metamorphosis. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 35–42, 1992.



- [24] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of ACM*, 9(1):61–63, 1962.
- [25] D. Benson and J. Davis. Octree textures. *ACM Transactions on Graphics*, 21(3):785–790, 2002.
- [26] D. F. Berman, J. T. Bartell, and D. H. Salesin. Multiresolution painting and compositing. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 85–90, 1994.
- [27] P. Bhat, C. L. Zitnick, N. Snavely, A. Agarwala, M. Agrawala, B. Curless, M. Cohen, and S. B. Kang. Using photographs to enhance videos of a static scene. In *Eurographics Symposium on Rendering*, pages 327–338, 2007.
- [28] A. Blake and M. Isard. *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer-Verlag, 1998. ISBN 978-3540762171.
- [29] D. Blostein and N. Ahuja. Shape from texture: Integrating texture-element extraction and surface estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11: 1233–1251, 1989.
- [30] D. Blythe, B. Grantham, M. J. Kilgard, T. McReynolds, and S. R. Nelson. Advanced graphics programming techniques using OpenGL. In *ACM SIGGRAPH 1999 courses*, ACM SIGGRAPH, 1999.
- [31] A. Bornik and A. Ferko. Texture minification using quad-trees and mipmaps. In *Eurographics 2002 Short Presentations*, pages 263–272, 2002.
- [32] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.
- [33] A. Bradley, F. Heide, and W. Heidrich. CALTag: High precision fiducial markers for camera calibration. In *Vision, Modeling and Visualization*, pages 41–48, 2010.
- [34] R. Brinkmann. *The Art and Science of Digital Compositing: Techniques for Visual Effects, Animation and Motion Graphics*. Morgan Kaufmann, 2nd edition, 2008. ISBN 978-0123706386.
- [35] T. Brox, A. Bruhn, N. Papenbergh, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision*, pages 25–36, 2004.

- [36] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured Lumigraph Rendering. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 425–432, 2001.
- [37] P. J. Burt and E. H. Adelson. A Multiresolution Spline With Application to Image Mosaics. *Computer Graphics*, 17(3):217–236, 1983.
- [38] J. Cameron. *Avatar*, 2009. J. Landau, Twentieth Century Fox.
- [39] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
- [40] J. Carranza, C. Theobalt, M. Magnor, and H. P. Seidel. Free-viewpoint video of human actors. *ACM Transaction on Graphics*, 22(3):569–577, 2003.
- [41] E.E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Departement of Computer Sciences, University of Utah, 1974.
- [42] J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong. Plenoptic Sampling. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 307–318, 2000.
- [43] H. Chang, D.-Y. Yeung, and Y.M. Xiong. Super-resolution through neighbor embedding. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 275–282, 2004.
- [44] S. E. Chen and L. Williams. View Interpolation for Image Synthesis. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 279–288, 1993.
- [45] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2001.
- [46] Y.-Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski. Video matting of complex scenes. *ACM Transactions on Graphics*, 21(3):243–248, 2002.
- [47] 360 Cities. 360 cities, 2011. <http://www.360cities.net>, visited in Feb. 2011.
- [48] D. Cobzaş, K. Yerex, and M. Jägersand. Dynamic textures for image-based rendering of fine-scale 3D structure and animation of non-rigid motion. *Computer Graphics Forum*, 21(3):493–502, 2002.

- [49] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. *ACM Transaction on Graphics*, 22(3):287–294, 2003.
- [50] R. T. Collins. A space-sweep approach to true multi-image matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 358–363, 1996.
- [51] M. Corsini, M. Dellepiane, F. Ponchio, and R. Scopigno. Image-to-geometry registration: a mutual information method exploiting illumination-related geometric properties. *Computer Graphics Forum*, 28(7):1755–1764, 2009.
- [52] F. C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18:207–212, 1984.
- [53] S. Dai, M. Han, W. Xu, Y. Wu, and Y. Gong. Soft edge smoothness prior for alpha channel super resolution. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
- [54] E. de Aguiar, C. Theobalt, M. Magnor, and H.-P. Seidel. Reconstructing human shape and motion from multi-view video. In *European Conference on Visual Media Production*, pages 42–49, 2005.
- [55] E. de Aguiar, C. Theobalt, C. Stoll, and H.-P. Seidel. Marker-less deformable mesh tracking for human shape and motion capture. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
- [56] E. de Aguiar, C. Theobalt, C. Stoll, and H.-P. Seidel. Rapid animation of laser-scanned humans. In *Virtual Reality*, pages 223–226, 2007.
- [57] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. *ACM Transactions on Graphics*, 27(3):1–10, 2008.
- [58] P. Debevec, Y. Yu, and G. Boshokov. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *Eurographics Rendering Workshop*, pages 105–116, 1998.
- [59] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 11–20, 1996.
- [60] D. DeBry, J. Gibbs, D. D. Petty, and N. Robins. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics*, 21(3):763–768, 2002.

- [61] B. De Decker and P. Beckaert. Interactive acquisition and rendering of human actors. In *Workshop on Content Generation and Coding for 3D-Television*, pages 1–4, 2006.
- [62] J. D. Durou, M. Falcone, and M. Sagona. Numerical methods for shape-from-shading: A new survey with benchmarks. *Computer Vision and Image Understanding*, 109(1): 22–43, 2008.
- [63] C. R. Dyer. Volumetric Scene Reconstruction from Multiple Views. In *Foundations of Image Understanding*, pages 469–489, 2001.
- [64] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *ACM Transactions on Graphics*, 20(3): 341–346, 2001.
- [65] A. A. Efros and Thomas K. Leung. Texture Synthesis by Non-parametric Sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, 1999.
- [66] P. Einarsson, C.-F. Chabert, A. Jones, W.-C. Ma, B. Lamond, T. Hawkins, M. Bolas, S. Sylwan, and P. Debevec. Relighting Human Locomotion with Flowed Reflectance Fields. In *Eurographics Symposium on Rendering*, pages 183–194, 2006.
- [67] M. Eisemann and M. Magnor. Filtered Blending and Floating Textures: Ghosting-free Projective Texturing with Multiple Images, TR 2007-5-3. Technical report, Computer Graphics Lab, TU Braunschweig, 2007.
- [68] M. Eisemann and M. Magnor. ZIPMAPs: Zoom-into-parts texture maps, TR 2008-11-8. Technical report, Computer Graphics Lab, TU Braunschweig, 2008.
- [69] M. Eisemann and M. Magnor. ZIPMAPs: Zoom-Into-Parts Texture Maps. In *Vision, Modeling, and Visualization*, pages 291–297, 2010.
- [70] M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating Textures, TR 2008-10-4. Technical report, Computer Graphics Lab, TU Braunschweig, 2007.
- [71] M. Eisemann, A. Sellent, and M. Magnor. Filtered Blending: A new, minimal Reconstruction Filter for Ghosting-Free Projective Texturing with Multiple Images. In *Vision, Modeling, and Visualization*, pages 119–126, 2007.
- [72] M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent.

- Floating Textures. *Computer Graphics Forum*, 27(2):409–418, 2008.
- [73] M. Eisemann, J. Wolf, and M. Magnor. Spectral Video Matting. In *Vision, Modeling, and Visualization*, pages 121–126, 2009.
- [74] M. Eisemann, E. Eisemann, H.-P. Seidel, and M. Magnor. Photo zoom: High resolution from unordered image collections. In *Graphics Interface*, pages 71–78, 2010.
- [75] M. Eisemann, D. Gohlke, and M. Magnor. Structure-Aware Image Compositing, TR 2010-11-12. Technical report, Computer Graphics Lab, TU Braunschweig, 2010.
- [76] M. Eisemann, T. Stich, and M. Magnor. 3-D Cinematography with Approximate or No Geometry. In Ronfard and Taubin [200], pages 259–284. ISBN 978-3642123917.
- [77] M. Eisemann, D. Gohlke, and M. Magnor. Edge-constrained image compositing. In *Graphics Interface*, 2011. Accepted for publication.
- [78] R. Fattal. Image upsampling via imposed edge statistics. *ACM Transactions on Graphics*, 26(3):1–8, 2007.
- [79] P. Felzenszwalb and D. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59:167–181, 2004.
- [80] A. Finkelstein, C. E. Jacobs, and D. H. Salesin. Multiresolution video. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 281–290, 1996.
- [81] M.A. Fischler and R. Bolles. Random Sample Consensus. A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [82] A. Fitzgibbon, Y. Wexler, and A. Zisserman. Image-based rendering using image-based priors. *International Journal of Computer Vision*, 63(2):141–151, 2005.
- [83] Graham Flint. Gigapxl project, 2011. <http://www.gigapxl.org>, visited in Feb. 2011.
- [84] J.-S. Franco and E. Boyer. Exact polyhedral visual hulls. In *British Machine Vision Conference*, pages 329–338, 2003.
- [85] W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, 2002.

- [86] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Towards Internet-scale multi-view stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1434–1441, 2010.
- [87] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22, 2000.
- [88] R. Gal, Y. Wexler, E. Ofek, H. Hoppe, and D. Cohen-Or. Seamless montage for texturing models. *Computer Graphics Forum*, 29(2):479–486, 2010.
- [89] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
- [90] E. S. L. Gastal and M. M. Oliveira. Shared sampling for real-time alpha matting. *Computer Graphics Forum*, 29(2):575–584, 2010.
- [91] M. Germann, A. Hornung, R. Keiser, R. Ziegler, S. Wurmlin, and M. Gross. Articulated billboards for video-based rendering. *Computer Graphics Forum*, 29(2):585–594, 2010.
- [92] D. Glasner, S. Bagon, and M. Irani. Super-resolution from a single image. In *IEEE International Conference on Computer Vision*, 2009.
- [93] M. Gleicher. Image snapping. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 183–190, 1995.
- [94] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz. Multi-view stereo for community photo collections. In *IEEE International Conference on Computer Vision*, pages 265–270, 2007.
- [95] M. Goesele, J. Ackermann, S. Fuhrmann, C. Haubold, R. Klowsky, D. Steedly, and R. Szeliski. Ambient point clouds for view interpolation. *ACM Transactions on Graphics*, 29(3):1–6, 2010.
- [96] D. B. Goldman, B. Curless, A. Hertzmann, and S. M. Seitz. Shape and Spatially-Varying BRDFs from Photometric Stereo. In *IEEE International Conference on Computer Vision*, pages 341–348, 2005.
- [97] M. Gong, L. Wang, R. Yang, and Y.-H. Yang. Real-time video matting using multichannel poisson equations. In *Graphics Interface*, pages 89–96, 2010.

- [98] Google. Google earth, 2011. <http://www.google.com/earth/index.html>, visited in Feb. 2011.
- [99] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 43–54, 1996.
- [100] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann. Random walks for interactive alpha-matting. In *IASTED International Conference on Visualization, Imaging and Image Processing*, pages 423–429, 2005.
- [101] N. Greene and P. S. Heckbert. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications*, 6:21–27, 1986.
- [102] X. Gu, S. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.
- [103] Y. Guan, W. Chen, X. Liang, Z. Ding, and Q. Peng. Easy matting - a stroke based approach for continuous image matting. *Computer Graphics Forum*, 25(3):567–576, 2006.
- [104] A. Gupta, P. Bhat, M. Dontcheva, B. Curless, O. Deussen, and M. Cohen. Enhancing and experiencing spacetime resolution with videos and stills. In *International Conference on Computational Photography*, pages 1–9, 2009.
- [105] Y. HaCohen, R. Fattal, and D. Lischinski. Image upsampling via texture hallucination. In *International Conference on Computational Photography*, pages 1–8, 2010.
- [106] C. Han and H. Hoppe. Optimizing continuity in multiscale imagery. *ACM Transactions on Graphics*, 29(5):1–9, 2010.
- [107] C. Han, E. Risser, R. Ramamoorthi, and E. Grinspun. Multiscale texture synthesis. *ACM Transactions on Graphics*, 27(3):1–8, 2008.
- [108] J. Han, K. Zhou, L.-Y. Wei, M. Gong, H. Bao, X. Zhang, and B. Guo. Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer: International Journal of Computer Graphics*, 22(9):918–925, 2006.
- [109] M.J. Hannah. *Computer matching of areas in stereo images*. PhD thesis, Stanford University, 1974.
- [110] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2006. ISBN 978-0521540513.

- [111] N. Hasler, C. Stoll, M. Sunkel, B. Rosenhahn, and H.-P. Seidel. A statistical model of human pose and body shape. *Computer Graphics Forum*, 2(28):337–346, 2009.
- [112] J. Hays and A. A. Efros. Scene completion using millions of photographs. In *ACM Transaction on Graphics*, pages 1–8, 2007.
- [113] P.S. Heckbert. Survey of texture mapping. In *Graphics Interface / Vision Interface*, pages 207–212, 1986.
- [114] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. *ACM Transactions on Graphics*, 20(3):327–340, 2001.
- [115] A. Hilton, J.-Y. Guillemaut, J. Kilner, O. Grau, and G. Thomas. Free-Viewpoint Video for TV Sport Production. In Ronfard and Taubin [200], pages 77–106. ISBN 978-3642123917.
- [116] B.K.P. Horn and B.G. Schunck. Determining Optical Flow. *Artificial Intelligence*, 16:185–203, 1981.
- [117] A. Hornung and L. Kobbelt. Interactive pixel-accurate free viewpoint rendering from images with silhouette aware sampling. *Computer Graphics Forum*, 28(8):2090–2103, 2009.
- [118] J. Hoschek and D. Lasser. *Fundamentals of computer aided geometric design*. A. K. Peters, Ltd., 1993. ISBN 1-56881-007-5.
- [119] Y. Huang and X. Zhuang. Motion-partitioned adaptive block matching for video compression. In *International Conference on Image Processing*, pages 554–562, 1995.
- [120] T. Hüttner. High resolution textures. In *IEEE Visualization*, pages 13–17, 1998.
- [121] A. Isaksen, L. McMillan, and S. J. Gortler. Dynamically Reparameterized Light Fields. *ACM Transactions on Graphics*, 19(3):297–306, 2000.
- [122] R. M. Ismert, K. Bala, and D. P. Greenberg. Detail synthesis for image-based texturing. In *Symposium on Interactive 3D graphics*, pages 171–175, 2003.
- [123] S. Jeschke, D. Cline, and P. Wonka. Rendering surface details with diffusion curves. *ACM Transaction on Graphics*, 28(5):1–8, 2009.
- [124] J. Jia and C. Tang. Eliminating Structure and Intensity Misalignment in Image Stitching. In *IEEE International Conference on Computer Vision*, pages 1651–1658, 2005.



- [125] J. Jia and C.-K. Tang. Image stitching using structure deformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(4):617–631, 2008.
- [126] J. Jia, J. Sun, C. Tang, and H. Shum. Drag-and-Drop Pasting. *ACM Transactions on Graphics*, 25(5):631–637, 2006.
- [127] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 2nd edition, 2002. ISBN 978-0387954424.
- [128] N. Joshi, W. Matusik, and S. Avidan. Natural video matting using camera arrays. *ACM Transactions on Graphics*, 25(3):779–786, 2006.
- [129] D. Kelly. *Digital Compositing in Depth*. The Coriolis Group, 2000. ISBN 978-1576104316.
- [130] K. I. Kim and Y. Kwon. Example-based learning for single-image super-resolution. In *Symposium of the German Association for Pattern Recognition*, pages 456–463, 2008.
- [131] J. Kopf, M. Uyttendaele, O. Deussen, and M. F. Cohen. Capturing and viewing gigapixel images. *ACM Transactions on Graphics*, 26(3):1–10, 2007.
- [132] M. Kraus and T. Ertl. Adaptive texture maps. In *ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 7–15, 2002.
- [133] K. N. Kutulakos and S. M. Seitz. A Theory of Shape by Space Carving. *International Journal on Computer Vision*, 38(3):199–218, 2000.
- [134] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.
- [135] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics*, 24(3):795–802, 2005.
- [136] J. Lacoste, T. Boubekur, B. Jobard, and C. Schlick. Appearance preserving octree-textures. In *GRAPHITE*, pages 87–93, 2007.
- [137] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, and M. Zwicker. State of the art in procedural noise functions. In *Eurographics 2010 - State of the Art Reports*, 2010.
- [138] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

- [139] S. Lefebvre and H. Hoppe. Parallel controllable texture synthesis. *ACM Transactions on Graphics*, 24(3):777–786, 2005.
- [140] S. Lefebvre and H. Hoppe. Compressed random-access trees for spatially coherent data. In *Rendering Techniques*, pages 339–349, 2007.
- [141] S. Lefebvre, S. Hornus, and F. Neyret. Texture sprites: Texture elements splatted on surfaces. In *Symposium on Interactive 3D Graphics*, pages 163–170, 2005.
- [142] H. P. A. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.-P. Seidel. Image-based reconstruction of spatial appearance and geometric detail. *ACM Transactions on Graphics*, 22(2):234–257, 2003.
- [143] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless Image Stitching in the Gradient Domain. In *European Conference on Computer Vision*, volume 4, pages 377–389, 2003.
- [144] A. Levin, A. Rav-Acha, and D. Lischinski. Spectral matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
- [145] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242, 2008.
- [146] M. Levoy and P. Hanrahan. Light Field Rendering. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 31–42, 1996.
- [147] M. Li, M. Magnor, and H.-P. Seidel. Hardware-accelerated rendering of photo hulls. *Computer Graphics Forum*, 23(3):635–642, 2004.
- [148] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. In *ACM Transactions on Graphics*, pages 595–600, 2005.
- [149] Z. Lin and H.-Y. Shum. On the Number of Samples Needed in Light Field Rendering with constant-depth assumption. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 579–588, 2000.
- [150] Z. Lin and H.-Y. Shum. A Geometric Analysis of Light Field Rendering. *International Journal of Computer Vision*, 58(2):121–138, 2004.
- [151] C. Linz, C. Lipski, and M. Magnor. Multi-image Interpolation based on Graph-Cuts and Symmetric Optic Flow. In *Vision, Modeling and Visualization*, pages 115–122, 2010.

- [152] C. Lipski, D. Bose, M. Eisemann, K. Berger, and M. Magnor. Sparse bundle adjustment speedup strategies. In *International Conference on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 85–88, 2010.
- [153] C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor. Virtual video camera: Image-based viewpoint navigation through space and time. *Computer Graphics Forum*, 29(8): 2555–2568, 2010.
- [154] C. Lipski, C. Linz, T. Neumann, and M. Magnor. High resolution image correspondences for video post-production. In *Conference on Visual Media Production*, pages 33–39, 2010.
- [155] X. Liu, L. Wan, Y. Qu, T.-T. Wong, S. Lin, C.-S. Leung, and P.-A. Heng. Intrinsic colorization. *ACM Transactions on Graphics*, 27(5):1–9, 2008.
- [156] Y. Liu, G. Chen, N. Max, C. Hofsetz, and P. McGuiness. Undersampled Light Field Rendering by a Plane Sweep. *Computer Graphics Forum*, 25(2):225–236, 2006.
- [157] F. Losasso. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics*, 23(3): 769–776, 2004.
- [158] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [159] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.
- [160] M. Magnor, M. Pollefeys, G. Cheung, W. Matusik, and C. Theobalt. Video-based rendering. In *ACM SIGGRAPH 2005 Courses*, ACM SIGGRAPH, 2005.
- [161] D. Mahajan, F.-C. Huang, W. Matusik, R. Ramamoorthi, and P. Belhumeur. Moving Gradients: A Path-Based Method for Plausible Image Interpolation. *ACM Transactions on Graphics*, 28(3):1–10, 2009.
- [162] J. Maintz and M. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.
- [163] R. A. Manning and C. R. Dyer. Interpolating view and scene motion by dynamic view morphing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 388–394, 1999.

- [164] W. Mark, L. McMillan, and G. Bishop. Post-Rendering 3D Warping. In *Symposium on Interactive 3D Graphics*, pages 7–16, 1997.
- [165] W. Matusik and H. Pfister. 3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *ACM Transactions on Graphics*, 23(3):814–824, 2004.
- [166] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 369–374, 2000.
- [167] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Eurographics Workshop on Rendering*, pages 115–125, 2001.
- [168] W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3D photography using opacity hulls. *ACM Transactions on Graphics*, 21(3):427–437, 2002.
- [169] H. Mayer, A. Bornik, J. Bauer, K. Karner, and F. Leberl. Multiresolution texture for photorealistic rendering. In *IEEE Spring Conference on Computer Graphics*, pages 174–183, 2001.
- [170] M. McGuire, W. Matusik, H. Pfister, J. F. Hughes, and Frédo Durand. Defocus video matting. *ACM Transactions on Graphics*, 24(3):567–576, 2005.
- [171] M. McGuire, W. Matusik, and W. Yezaur. Practical, Real-time Studio Matting using Dual Imagers. In *Eurographics Symposium on Rendering*, pages 235–244, 2006.
- [172] J. Meinguet. Multivariate interpolation at arbitrary points made simple. *Journal of Applied Mathematics and Physics*, 5: 439–468, 1979.
- [173] D. P. Mitchell and A. N. Netravali. Reconstruction Filters in Computer-Graphics. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 221–228, 1988.
- [174] M. Mittring and Crytek GmbH. Advanced virtual texture topics. In *ACM SIGGRAPH 2008 courses*, ACM SIGGRAPH, pages 23–51, 2008.
- [175] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *Conference on Computer graphics*

- and interactive techniques, ACM SIGGRAPH, pages 191–198, 1995.
- [176] G. Müller, J. Meseth, M. Sattler, R. Sarlette, and R. Klein. Acquisition, synthesis and rendering of bidirectional texture functions. In *Eurographics 2004, State of the Art Reports*, pages 69–94, 2004.
- [177] T. Naemura, J. Tago, and H. Harashima. Real-time video-based modeling and rendering of 3D scenes. *Computer Graphics and Applications*, 22(2):66–73, 2002.
- [178] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2001.
- [179] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, 2006.
- [180] E. Ofek, E. Shilat, A. Rappoport, and M. Werman. Multiresolution textures from image sequences. *IEEE Computer Graphics and Applications*, 17(2):18–29, 1997.
- [181] E. Ofek, E. Shilat, and M. Werman. Highlight and reflection-independent multiresolution textures from image sequences. *IEEE Computer Graphics and Applications*, 17:18–29, 1997.
- [182] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.
- [183] P. Panareda Busto, C. Eisenacher, S. Lefebvre, and M. Stamminger. Instant Texture Synthesis by Numbers. *Vision, Modeling and Visualization 2010*, pages 81–85, 2010.
- [184] P. Pérez, M. Gangnet, and A. Blake. Poisson Image Editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.
- [185] K. Perlin and L. Velho. Live paint: painting with procedural multiscale textures. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 153–160, 1995.
- [186] M. Piccardi. Background subtraction techniques: a review. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3099–3104, 2004.
- [187] T. Pock, M. Urschler, C. Zach, R. Beichel, and H. Bischof. A Duality Based Algorithm for TV-L<sub>1</sub>-Optical-Flow Image

- Registration. In *International Conference on Medical Image Computing and Computer Assisted Intervention*, pages 511–518, 2007.
- [188] D. Porquet, J.-M. Dischler, and D. Ghazanfarpour. Real-Time High-Quality View-Dependent Texture Mapping using Per-Pixel Visibility. In *GRAPHITE*, pages 213–220, 2005.
- [189] T. Porter and T. Duff. Compositing digital images. *Computer Graphics*, 18(3):253–259, 1984.
- [190] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007. ISBN 978-0521880688.
- [191] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-Based Rendering: Visualizing Real Objects from Scanned Range and Color Data. In *Eurographics Workshop on Rendering*, pages 23–34, 1997.
- [192] Z. Qin, M. D. McCool, and C. S. Kaplan. Real-time texture-mapped vector glyphs. In *Symposium on Interactive 3D graphics and games*, pages 125–132, 2006.
- [193] Z. Qin, M. D. McCool, and C. Kaplan. Precise vector textures for real-time 3D rendering. In *Symposium on Interactive 3D graphics and games*, pages 199–206, 2008.
- [194] G. Ramanarayanan and K. Bala. Constrained texture synthesis via energy minimization. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):167–178, 2007.
- [195] N. Ray, V. Nivoliere, S. Lefebvre, and B. Lévy. Invisible seams. *Computer Graphics Forum*, 29(4):1489–1496, 2010.
- [196] A. Reche-Martinez and G. Drettakis. View-Dependent Layered Projective Texture Maps. In *Pacific Conference on Computer Graphics and Applications*, pages 492–500, 2003.
- [197] X. Ren and J. Malik. Learning a classification model for segmentation. In *IEEE International Conference on Computer Vision*, pages 10–17, 2003.
- [198] Microsoft Research. Microsoft Photosynth, 2011. <http://photosynth.net>, visited in Feb. 2011.
- [199] C. Rocchini, P. Cignomi, C. Montani, and R. Scopigno. Multiple textures stitching and blending on 3D objects. In *Eurographics Workshop on Rendering*, pages 119–130, 1999.

- [200] R. Ronfard and G. Taubin, editors. *Image and Geometry Processing for 3-D Cinematography*. Springer-Verlag, 2010. ISBN 978-3642123917.
- [201] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut" - Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.
- [202] P. Sand and S. Teller. Particle Video: Long-Range Motion Estimation using Point Trajectories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2195–2202, 2006.
- [203] M. Sarim, A. Hilton, J.-Y. Guillemaut, H. Kim, and T. Takai. Multiple view wide-baseline trimap propagation for natural video matting. In *Conference on Visual Media Production*, pages 82–91, 2010.
- [204] H. S. Sawhney. Simplifying motion and structure analysis using planar parallax and image warping. In *International Conference on Pattern Recognition*, pages 403 – 408, 1994.
- [205] T. Schoenemann and D. Cremers. High resolution motion layer decomposition using dual-space graph cuts. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1 –7, 2008.
- [206] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects using Texture Mapping. *Computer Graphics*, 26(2):249–252, 1992.
- [207] S. Seitz and C. Dyer. Physically-valid view synthesis by image interplation. In *IEEE Workshop on Representation of Visual Scenes*, pages 18–26, 1995.
- [208] S. Seitz and C. Dyer. View Morphing. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 21–30, 1996.
- [209] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, 1999.
- [210] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 519–528, 2006.
- [211] A. Sellent, M. Eisemann, B. Goldlücke, D. Cremers, and M. Magnor. Motion field estimation from alternate exposure images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(PrePrints), 2010.

- [212] P. Sen. Silhouette maps for improved texture magnification. In *ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 65–73, 2004.
- [213] Peter Shirley. *Fundamentals of Computer Graphics 2nd Edition*. Transatlantic Publishers, 2005. ISBN 978-1568812694.
- [214] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 1.4*. Addison-Wesley Professional, 2003. ISBN 978-0321173485.
- [215] J. Sivic, B. Kaneva, A. Torralba, S. Avidan, and W. T. Freeman. Creating and exploring a large photorealistic virtual space. In *IEEE Workshop on Internet Vision*, pages 1391–1407, 2008.
- [216] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. A survey of methods for volumetric scene reconstruction from photographs. In *International WS on Volume Graphics*, pages 81–100, 2001.
- [217] P.-P. J. Sloan, M. F. Cohen, and S. J. Gortler. Time Critical Lumigraph Rendering. In *Symposium on Interactive 3D Graphics*, pages 17–24, 1997.
- [218] A. R. Smith and J. F. Blinn. Blue screen matting. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 259–268, 1996.
- [219] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3D. *ACM Transactions on Graphics*, 25(3):835–846, 2006.
- [220] N. Snavely, R. Garg, S. M. Seitz, and R. Szeliski. Finding paths through the world’s photos. *ACM Transactions on Graphics*, 27(3):11–21, 2008.
- [221] J. Starck and A. Hilton. Spherical matching for temporal correspondence of non-rigid surfaces. In *IEEE International Conference on Computer Vision*, pages 1387–1394, 2005.
- [222] J. Starck and A. Hilton. Virtual view synthesis of people from multiple view video sequences. *Graphical Models*, 67(6):600–620, 2005.
- [223] J. Starck and A. Hilton. Surface capture for performance based animation. *IEEE Computer Graphics and Applications*, 27(3):21–31, 2007.
- [224] F. Steinbrücker, T. Pock, and D. Cremers. Advanced data terms for variational optic flow estimation. In *Vision, Modeling, and Visualization*, pages 155–164, 2009.



- [225] F. Steinbruecker, T. Pock, and D. Cremers. Large displacement optical flow computation without warping. In *IEEE International Conference on Computer Vision*, pages 1609–1614, 2009.
- [226] J. Stewart, J. Yu, S. J. Gortler, and L. McMillan. A New Reconstruction Filter for Undersampled Light Fields. In *Eurographics Workshop on Rendering*, pages 150–156, 2003.
- [227] T. Stich. *Space-Time Interpolation Techniques*. PhD thesis, Computer Graphics Lab, TU Braunschweig, Germany, 2009.
- [228] T. Stich, C. Linz, G. Albuquerque, and M. Magnor. View and Time Interpolation in Image Space. *Computer Graphics Forum*, 27(7):1781–1787, 2008.
- [229] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor. Perception-motivated Interpolation of Image Sequences. In *Symposium on Applied Perception in Graphics and Visualization*, pages 97–106, 2008.
- [230] J. Sun, N.-N. Zheng, H. Tao, and H.-Y. Shum. Image hallucination with primal sketch priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 729–736, 2003.
- [231] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson matting. *ACM Transaction on Graphics*, 23(3):315–321, 2004.
- [232] J. Sun, Z.B. Xu, and H.Y. Shum. Image super-resolution using gradient profile prior. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.
- [233] J. Sun, J. Zhu, and M. F. Tappen. Context-constrained hallucination for image super-resolution. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 231–238, 2010.
- [234] Kalyan Sunkavalli, Micah K. Johnson, Wojciech Matusik, and Hanspeter Pfister. Multi-scale image harmonization. *ACM Transactions on Graphics*, 29(3):1–10, 2010.
- [235] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, 1996.
- [236] R. Szeliski. Image alignment and stitching: a tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–104, 2006.

- [237] T. Takai, A. Hilton, and T. Matsuyama. Harmonized Texture Mapping. In *International Symposium on 3D Data Processing, Visualization and Transmission*, pages 1–8, 2010.
- [238] C. C. Tanner, C. J. Migdal, and M. T. Jones. The clipmap: a virtual mipmap. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 151–158, 1998.
- [239] M. Tarini and P. Cignoni. Pinchmaps: Textures with customizable discontinuities. *Computer Graphics Forum*, 24(3): 557–568, 2005.
- [240] Christian Theobalt, Stephan Wuermlin, Edilson de Aguiar, and Christoph Niederberger. *New Trends in 3D Video*. Eurographics Tutorial, 2007.
- [241] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, pages 839–846, 1998.
- [242] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics*, 21(3):665–672, 2002.
- [243] B. Triggs, P. F. McLauchlan, R. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *International Workshop on Vision Algorithms: Theory and Practice*, pages 298–372, 2000.
- [244] R.Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.
- [245] J. Tumblin and P. Choudhury. Bixels: Picture samples with sharp embedded boundaries. In *Rendering Techniques*, pages 255–264, 2004.
- [246] T. Tung, S. Nobuhara, and T. Matsuyama. Simultaneous super-resolution and 3D video using graph-cuts. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.
- [247] Y. Tzur and A. Tal. FlexiStickers - Photogrammetric Texture Mapping using Casual Images. *ACM Transactions on Graphics*, 28(3):1–10, 2009.
- [248] M. Uyttendaele, A. Eden, and R. Szeliski. Eliminating Ghosting and Exposure Artifacts in Image Mosaics. In

- IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 509–516, 2001.
- [249] J. D. van Ouwerkerk. Image super-resolution survey. *Image Vision Computing*, 24(10):1039–1052, 2006.
- [250] M. Vasilescu, Alex O., and D. Terzopoulos. TensorTextures: multilinear image-based rendering. *ACM Transactions on Graphics*, 23(3):336–342, 2004.
- [251] S. Vedula, S. Baker, and T. Kanade. Image based spatio-temporal modeling and view interpolation of dynamic events. *ACM Transactions on Graphics*, 24(2):240–261, 2005.
- [252] J. Wang and M. F. Cohen. An iterative optimization approach for unified image segmentation and matting. In *IEEE International Conference on Computer Vision*, pages 936–943, 2005.
- [253] J. Wang and M. F. Cohen. Image and video matting: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(2):97–175, 2007.
- [254] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. *ACM Transactions on Graphics*, 24(3):585–594, 2005.
- [255] J. Wang, M. Agrawala, and M. F. Cohen. Soft scissors: an interactive tool for realtime high quality matting. *ACM Transactions on Graphics*, 26(3):1–9, 2007.
- [256] L. Wang and K. Mueller. Generating sub-resolution detail in images and volumes using constrained texture synthesis. In *IEEE Conference on Visualization*, pages 75–82, 2004.
- [257] Z. Wang and E. P. Simoncelli. Translation insensitive image similarity in complex wavelet domain. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 573–576, 2005.
- [258] M. Waschbüsch, S. Würmlin, and M. H. Gross. 3D Video Billboard Clouds. *Computer Graphics Forum*, 26(3):561–569, 2007.
- [259] L.-Y. Wei. Tile-based texture mapping on graphics hardware. In *SIGGRAPH 2004 Sketches*, pages 67–74, 2004.
- [260] L. Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 479–488, 2000.

- [261] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report*, pages 1–25, 2009.
- [262] Y. Weiss. Segmentation using eigenvectors: A unifying view. In *International Conference on Computer Vision*, pages 975–982, 1999.
- [263] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *British Machine Vision Conference*, pages 1–11, 2009.
- [264] O. Whyte, J. Sivic, and A. Zisserman. Get Out of my Picture! Internet-based Inpainting. In *British Machine Vision Conference*, pages 1–11, 2009.
- [265] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. Horowitz. The Light Field Video Camera. In *Media Processors*, pages 1–8, 2002.
- [266] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High performance imaging using large camera arrays. *ACM Transactions on Graphics*, 24(3):765–776, 2005.
- [267] L. Williams. Casting curved shadows on curved surfaces. In *Computer Graphics*, volume 12, pages 270–274, 1978.
- [268] L. Williams. Pyramidal parametrics. *ACM SIGGRAPH Computer Graphics*, 17(3):1–11, 1983.
- [269] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1990. ISBN 978-0818689444.
- [270] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle. Surface light fields for 3D photography. In *Conference on Computer graphics and interactive techniques*, ACM SIGGRAPH, pages 287–296, 2000.
- [271] O. Woodford and A. W. Fitzgibbon. Fast image-based rendering using hierarchical image-based priors. In *British Machine Vision Conference*, volume 1, pages 260–269, 2005.
- [272] Q. Wu and Y. Yu. Feature Matching and Deformation for Texture Synthesis. *ACM Transactions on Graphics*, 23(3):364–367, 2004.
- [273] J. Xiao, C. Rao, and M. Shah. View Interpolation for Dynamic Scenes. In *European Conference on Computer Graphics*, pages 153–162, 2002.

- [274] Yahoo. Flickr, 2011. <http://www.flickr.com>, visited in Feb. 2011.
- [275] J.C. Yang, J. Wright, T.S. Huang, and Y. Ma. Image super-resolution as sparse representation of raw image patches. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.
- [276] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 211–220, 2003.
- [277] S. X. Yu and J. Shi. Multiclass spectral clustering. In *IEEE International Conference on Computer Vision*, volume 2, pages 313–320, 2003.
- [278] T. Yu, C. Zhang, M. Cohen, Y. Rui, and Y. Wu. Monocular video foreground/background segmentation by tracking spatial-color gaussian mixture models. In *IEEE Workshop on Motion and Video Computing*, pages 5–12, 2007.
- [279] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime TV-L<sup>1</sup> optical flow. In *DAGM conference on Pattern recognition*, pages 214–223, 2007.
- [280] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.
- [281] C. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics*, 23(3):600–608, 2004.
- [282] M. Zwicker, W. Matusik, F. Durand, and H. Pfister. Antialiasing for Automultiscopic 3D Displays. In *Eurographics Symposium on Rendering*, pages 107–114, 2006.



## CURRICULUM VITÆ - LEBENSLAUF

---

### CURRICULUM VITÆ

---

1980	born in Cologne, Germany
1999	Highschool degree, main subjects mathematics and biology Georg-Büchner Gymnasium, Cologne, Germany
2000 - 2006	Diploma in Computer Science Universität Koblenz-Landau, Germany
since 2006	Ph.D. Student Computer Science, Institut für Computergraphik TU Braunschweig, Germany

---

### LEBENSLAUF

---

1980	geboren in Köln
1999	Allgemeine Hochschulreife Georg-Büchner Gymnasium, Köln
2000 - 2006	Diplom Informatik Universität Koblenz-Landau
seit 2006	Wissenschaftlicher Mitarbeiter, Institut für Computergraphik TU Braunschweig

---





## PUBLICATIONS

---

### JOURNAL ARTICLES

- A. Tatu, G. Albuquerque, M. Eisemann, P. Bak, H. Theisel, M. Magnor, and D. Keim. Automated Analytical Methods to Support Visual Exploration of High-Dimensional Data. *IEEE Transactions on Visualization and Computer Graphics*, accepted for publication.
- A. Sellent, M. Eisemann, B. Goldlücke, D. Cremers, and M. Magnor. Motion Field Estimation from Alternate Exposure Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, accepted for publication.
- A. Sellent, M. Eisemann, M. Magnor. Robust Feature Matching in General Multi-Image Setups. *Journal of WSCG*, 19(1–3):1–8, 2011.
- D. J. Lehmann, G. Albuquerque, M. Eisemann, A. Tatu, H. Schumann, M. Magnor, and H. Theisel. Visualisierung und Analyse multidimensionaler Datensätze. *Informatik-Spektrum*, 33(5):589–600, 2010.
- M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating Textures. *Computer Graphics Forum*, 27(2):409–418, 2008. Received the Best Student Paper Award at Eurographics 2008
- M. Eisemann, T. Grosch, S. Müller, and M. Magnor. Fast Ray/Axis-Aligned Bounding Box Overlap Tests using Ray Slopes. *Journal of Graphic Tools*, 12(4):35–46, 2007.

### REFEREED CONFERENCE PAPERS

- M. Eisemann, D. Gohlke, M. Magnor. Edge-Constrained Image Compositing. In *Graphics Interface*, 2011, accepted for publication.
- M. Eisemann, G. Albuquerque, M. Magnor. Data Driven Color Mapping. In *EuroVA*, 2011, accepted for publication.
- M. Eisemann, E. Eisemann, H.-P. Seidel, M. Magnor. Photo Zoom: High Resolution from Unordered Image Collections. In *Graphics Interface*, pages 71–78, 2010.

- P. Bauszat, M. Eisemann, and M. Magnor. The Minimal Bounding Volume Hierarchy. In *Vision, Modeling and Visualization*, pages 227–234, 2010.
- M. Eisemann and M. Magnor. ZIPMAPS: Zoom-Into-Parts Texture Maps. In *Vision, Modeling and Visualization*, pages 291–297, 2010.
- G. Albuquerque, M. Eisemann, D. J. Lehmann, H. Theisel, and M. Magnor. Improving the Visual Analysis of High-dimensional Datasets Using Quality Measures. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 19–26, 2010.
- C. Lipski, D. Bose, M. Eisemann, K. Berger, and M. Magnor. Sparse Bundle Adjustment Speedup Strategies. In *WSCG Communication Papers Proceedings*, pages 85–88, 2010.
- G. Albuquerque, M. Eisemann, D. J. Lehmann, H. Theisel, and M. Magnor. Quality-Based Visualization Matrices. In *Vision, Modeling and Visualization*, pages 341–349, 2009.
- M. Eisemann, J. Wolf, and M. Magnor. Spectral Video Matting. In *Vision, Modeling and Visualization*, pages 121–126, 2009.
- A. Sellent, M. Eisemann, B. Goldlücke, T. Pock, D. Cremers, and M. Magnor. Variational Optical Flow from Alternate Exposure Images. In *Vision, Modeling and Visualization*, pages 135–143, 2009.
- A. Tatu, G. Albuquerque, M. Eisemann, J. Schneidewind, H. Theisel, M. Magnor, and D. Keim. Combining automated analysis and visualization techniques for effective exploration of high-dimensional data. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 59–66, 2009.
- A. Sellent, M. Eisemann, and M. Magnor. Motion Field and Occlusion Time Estimation via Alternate Exposure Flow. In *IEEE International Conference on Computational Photography*, pages 1–8, 2009.
- M. Eisemann, C. Woizschke, and M. Magnor. Ray Tracing with the Single-Slab Hierarchy. In *Vision, Modeling and Visualization*, pages 373–381, 2008.
- M. Eisemann, A. Sellent, and M. Magnor. Filtered Blending: A new, minimal Reconstruction Filter for Ghosting-Free Projective Texturing with Multiple Images. In *Vision, Modeling and Visualization*, pages 119–126, 2007.

- M. Eisemann, T. Grosch, M. Magnor, and S. Müller. Automatic Creation of Object Hierarchies for Ray Tracing Dynamic Scenes. In *WSCG Short Communications Proceedings*, pages 57–64, 2007.

## BOOK CHAPTERS

- M. Eisemann, T. Stich, M. Magnor. 3-D Cinematography with Approximate or No Geometry. In *Image and Geometry Processing for 3-D Cinematography Rémi Ronfard, Gabriel Taubin, eds.*, Springer-Verlag, Berlin, Heidelberg, Germany, pages 259–284, 2010, ISBN 978-3642123917.

## BOOKS

- M. Eisemann. *Ray Tracing mittels dynamischer Bounding Volume Hierarchien: Eine Einführung und neue Methoden für dynamische Szenen*. VDM Verlag, 2008, ISBN 978-3836499101.

## TECHNICAL REPORTS

- M. Eisemann, D. Gohlke, and M. Magnor. Structure-Aware Image Compositing. Technical Report 2010-11-12, Computer Graphics Lab, TU Braunschweig, 2010.
- M. Eisemann and M. Magnor. ZIPMAPs: Zoom-into-parts texture maps. Technical Report 2008-11-8, Computer Graphics Lab, TU Braunschweig, 2008.
- A. Sellent, M. Eisemann, and M. Magnor. Calculating Motion Fields from Images with Two Different Exposure Times. Technical Report 2008-5-6, TU Braunschweig, 2008.
- M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating Textures. Technical Report 2008-10-4, Computer Graphics Lab, TU Braunschweig, 2008.
- M. Eisemann and M. Magnor. Filtered Blending and Floating Textures: Ghosting-free Projective Texturing with Multiple Images. Technical Report 2007-5-3, Computer Graphics Lab, TU Braunschweig, 2007.
- M. Eisemann, T. Grosch, M. Magnor, and S. Müller. Automatic Creation of Object Hierarchies for Ray Tracing Dynamic Scenes. Technical Report 2006-6-1, Computer Graphics Lab, TU Braunschweig, 2006.

## REFEREED POSTERS

- M. Eisemann, E. Eisemann, H.-P. Seidel, and M. Magnor. Photo Zoom: High Resolution from Unordered Image Collections. Poster at Siggraph, 2010.
- M. Eisemann, T. Grosch, M. Magnor, and S. Müller. Automatic Creation of Object Hierarchies for Ray Tracing Dynamic Scenes. Poster at Eurographics Symposium on Rendering, 2006.