Abstract of “Computational Videography with a Single Axis, Multi-Parameter Lens Camera” by Morgan McGuire, Ph.D., Brown University, May 2006.

What is the object in this picture? What would this object look like in a different scene? These questions are the core of computer vision (image analysis) and computer graphics (image synthesis). They are traditionally approached through specific problems like matting and composition, image segmentation, capture of high-speed video, depth-from-defocus, and visualization of multi-spectral video. For the solutions to these problems, both the inputs and outputs are frequently images. However, they are not the same kind of images as photographs, but a more general form of 2D maps over the surface of an object, lens, or display. For example, an image describing reflectivity over a surface is known as a texture map, an image composed of subimages with varying optical center is a light field slab, and an image describing the partial coverage of a backdrop by an object is a matte.

This dissertation introduces *multi-parameter video*, a framework for describing generalized 2D images over time that contain multiple samples at each pixel. Part of this framework is diagraming system called *optical splitting trees* for describing single-axis, multi-parameter, lens (SAMPL) camera systems that can be used to capture multi-parameter video. My thesis is that a SAMPL is a generic framework for graphics and vision data capture; that multi-parameter video can be accurately and efficiently captured by it; and that algorithms using this video as input can solve interesting graphics and vision problems. In defense of this thesis I demonstrate physical SAMPL camera hardware, many registered multi-parameter video streams captured with this system, and a series of problems solved using these videos. The leading problem I solve is the previously open unassisted video matting/subpixel object segmentation problem for complex, dynamic, and unknown backgrounds. To show generality I also demonstrate high speed video capture, multi-modal video fusion, multi-focus video fusion, and high dynamic range video capture.

Computational Videography with a Single Axis, Multi-Parameter Lens Camera

by

Morgan McGuire

B. S., Electrical Science and Computer Science, MIT, 2000

M. Eng., Computer Science and Electrical Engineering, MIT, 2000

Sc. M., Computer Science, Brown University, 2002

A dissertation submitted in partial fulfillment of the

requirements for the Degree of Doctor of Philosophy

in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2006

This dissertation by Morgan McGuire is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
John F. Hughes, Director

Recommended to the Graduate Council

Date _____ _____
Hanspeter Pfister, Reader

Date _____ _____
Michael Black, Reader

Approved by the Graduate Council

Date _____ _____
Karen Newman
Dean of the Graduate School

Abstract

What is the object in this picture? What would this object look like in a different scene? These questions are the core of computer vision (image analysis) and computer graphics (image synthesis). They are traditionally approached through specific problems like matting and composition, image segmentation, capture of high-speed video, depth-from-defocus, and visualization of multi-spectral video. For the solutions to these problems, both the inputs and outputs are frequently images. However, they are not the same kind of images as photographs, but a more general form of 2D maps over the surface of an object, lens, or display. For example, an image describing reflectivity over a surface is known as a texture map, an image composed of subimages with varying optical center is a light field slab, and an image describing the partial coverage of a backdrop by an object is a matte.

This dissertation introduces *multi-parameter video*, a framework for describing generalized 2D images over time that contain multiple samples at each pixel. Part of this framework is diagraming system called *optical splitting trees* for describing single-axis, multi-parameter, lens (SAMPL) camera systems that can be used to capture multi-parameter video. My thesis is that a SAMPL is a generic framework for graphics and vision data capture; that multi-parameter video can be accurately and efficiently captured by it; and that algorithms using this video as input can solve interesting graphics and vision problems. In defense of this thesis I demonstrate physical SAMPL camera hardware, many registered multi-parameter video streams captured with this system, and a series of problems solved using these videos. The leading problem I solve is the previously open unassisted video matting/subpixel object segmentation problem for complex, dynamic, and unknown backgrounds. To show generality I also demonstrate high speed video capture, multi-modal video fusion, multi-focus video fusion, and high dynamic range video capture.

Curriculum Vitae: M. McGuire

Education

- Candidate for *Ph.D. in Computer Science*, Brown University, Providence, RI.
  Advisor: John F. Hughes, Thesis: *Computational Videography with a Single Axis, Multi-Parameter Lens Camera*
- *Sc.M. in Computer Science*, Brown University, Providence, RI, May 2002.
  Advisor: John F. Hughes, Thesis: *Fast Shadow Volumes on Graphics Hardware*
- *M.Eng in Computer Science and Electrical Engineering*, MIT, Cambridge, MA, May 2000.
  Advisor: Stephen S. Ward, Thesis: *The Curl 2D Immediate Mode Graphics API*
- *B.S. in Electrical Science and Computer Science*, MIT, Cambridge, MA, May 2000.

Fellowships and Awards

- *1st place ACM SIGGRAPH Student Research Competition*, 2004.
- *International Graduate Fellowship*, NVIDIA, 2004.
- *Game Developer Magazine Front Line Award for Books*, 2004.
- *International Graduate Fellowship*, NVIDIA, 2003.
- *Andy van Dam Fellowship*, Brown University, 2002.
- *Hitachi America Scholarship*, Hitachi, 1994.

- *Navy Science Award*, U.S. Department of the Navy, 1994.

Teaching Experience

I assisted Hanspeter Pfister in advising George Stathis' Masters Thesis on Aspect Oriented Shading Trees at the Harvard Extension School, which won the best thesis award for all departments at that institution in 2005. I have advised several undergraduates: Pawel Wrotek, Andrea Fein, Colin Hartnett, Kevin Egan, Ethan Bromberg, and Alexander Rice through research publications at Brown University as part of our Games Research Group.

2006	Advisor to Games Course, (CS196) Brown
2004	Advisor to Group Independent Study on Games, Brown
2004	Instructor for Interactive Computer Graphics, (CS224) Brown
2002	Teaching Assistant for Interactive Computer Graphics, (CS224) Brown
1998–Present	Tutoring High-school students
1998–2000	Instructor for Beginner Sailing, MIT
1998	Instructor for High-school Studies Program Computer Science, MIT
1998	Teaching Assistant for High-school Studies Program Math, MIT
1996	Laboratory Assistant for Computational Structures, (6.004) MIT

Professional Appointments

2002–2004	Co-Founder and Chief Technical Officer at Blue Axion Games, Waltham, MA
2000–2002	Senior Architect at Oculus Technology Corporation, Boston, MA
1998–2000	Senior Graphics Architect at Curl Corporation, Cambridge, MA
1994–1998	Research Assistant at NEC Institute, Princeton, MA
1992–1994	Research Internship at IBM, Yorktown, NY

I have consulted since 1996. Clients include ROBLOX, Mitsubishi Electric Research Laboratory, LightSpace Technologies, BAE Systems, Valora Technologies, E Ink, and Pfizer.

I created and have managed the Open Source *G3D Engine* project since 2002.

Journal Papers

- Morgan McGuire, Wojciech Matusik, Hanspeter Pfister, John F. Hughes, and Frédo Durand. Defocus Video Matting. ACM Transaction on Graphics, July 2005 (SIGGRAPH 2005).
- Pawel Wrotek, Alexander Rice, and Morgan McGuire. Real-Time Bump Map Deformations using Graphics Hardware. to appear in jgt, 2005. *An early version of this work previously appeared as SIGGRAPH 2004 poster and took 2nd place in the ACM SIGGRAPH Student Research Competition.*
- Morgan McGuire. Observations on Silhouette Sizes. jgt, vol 9, no 1, 2004.
- Harold S. Stone, Bo Tao, and Morgan McGuire. Analysis of image registration noise due to rotationally dependent aliasing, J. Vis. Commun. Image. R. 14 (2003) pp. 114-135
- Morgan McGuire and Harold S. Stone. Techniques for Multiresolution Image Registration in the Presence of Occlusions. IEEE Transactions on Geoscience and Remote Sensing 38(3):1476-1479, May 2000 An early version of this paper appeared in Proceedings of Image Registration Workshop pp.101-122, 1997
- Harold S. Stone, J. LeMoigne, and Morgan McGuire. The Translation Sensitivity of Wavelet-Based Registration. IEEE Transactions on Pattern Analysis and Machine Intelligence 21(10):1074-1081, 1999 An early version of this paper in Proceedings of the 26th AIPR Workshop, Proceedings of the SPIE, Exploiting New Image Sources and Sensors 3240:116-125, 1997

Refereed Conference Papers, Posters, and SIGGRAPH Sketches

- M. McGuire and W. Matusik, Defocus Difference Matting, SIGGRAPH 2005 Sketch. Los Angeles, CA. August 2005
- S. Becker, S. Greenlee, D. Lemmerman, M. McGuire, N. Musurca, and N. Wardrip-Fruin, Cave Writing: Toward a Platform for Literary Immersive VR. SIGGRAPH 2005 Sketch. Los Angeles, CA, August 2005

- M. McGuire and M. McGuire, Steep Parallax Mapping, I3D 2005 Poster, Washington D.C, 2005

- M. McGuire and P. Sibley, A Heightfield on an Isometric Grid. SIGGRAPH 2004 Sketch. Los Angeles, CA. August 2004

- M. McGuire, A. Fein, and C. Hartnett, Real-Time Cartoon Smoke Rendering. SIGGRAPH 2004 Poster Abstract and Talk, Los Angeles, CA. 2004. *1st place 2004 ACM SIGGRAPH Student Research Competition*

- P. Wrotek, A. Rice, and M. McGuire. Real-Time Bump Map Deformations using Graphics Hardware, SIGGRAPH 2004 Poster Abstract and Talk. Los Angeles, CA, 2004* *2nd place 2004 ACM SIGGRAPH Student Research Competition*

- E. Bromberg-Martin, A. Jonsson, G. E. Marai, and M. McGuire, Hybrid Billboard Clouds for Model Simplification. SIGGRAPH 2004 Poster Abstract, Los Angeles, CA, 2004

- M. McGuire and J. F. Hughes, Hardware Contour Rendering. NPAR '04. Annecy, France. June 2004

- M. McGuire, S. Krishnamurthi, and J. F. Hughes, Programming Languages for Image Compression. ESOP 2002. Grenoble, France. June 2002

- H. S. Stone, J. Le Moigne, and M. McGuire, Image Registration Using Wavelet Techniques. Proc. of the 26th AIPR Workshop, Proc. of the SPIE, Exploiting New Image Sources and Sensors 3240:116-125, 1997*

- M. McGuire and H.S. Stone, Techniques for Multi-Resolution Image Registration in the Presence of Occlusions, Proc. of Image Registration Workshop pp.101-122, 1997

* significantly expanded versions of these projects later appeared in journals as listed above.

Invited Talks and Panels

- *New Results in Video Game Research*, UMass Dartmouth, North Dartmouth, MA, Sept. 24, 2004
- *Contours and Constants: Bounding NPR Rendering Time and Making it Fast on Graphics Hardware*, MIT, Cambridge, MA, August 2004
- *NPR and Terrain Rendering*, Harvard Extension School, Cambridge, MA, May 2004
- *MacVicar Panel on Education*, MIT, Cambridge, MA, February 10, 1998
- *The LASER Computer Graphics Language*, NEC USA, Princeton, NJ, August 1996
- *Principles of User Interface Design*, NEC USA, Princeton, NJ, August 1995

Books

- *Theory and Application of Computer Graphics Techniques*, Prentice Hall, NY, To be published 2006.
- The SuperShader, *in ShaderX*^{4}, W. Engel, to be published 2005.
- Effective Shadow Volume Rendering, *in GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, R. Fernando ed. Addison Wesley, 2004, ISBN: 0-321- 22832-4. *Won the 2004 Game Developer Magazine Front Line Award for Books.*

Other Works

- Copy-on-mutate optimizations for Matrices, *to appear in Dr. Dobb's Journal*, 2006
- The G3D 3D Engine, *to appear in Software 2.0*, September 2005
- Event Technical Wrap-Up: SIGGRAPH 2005. Gamasutra, Aug. 2005

- A. Vetro, M. McGuire, W. Matusik, A. Behrens, J. Lee, and H. Pfister, Test sequence for the MPEG multi-view working group, ISO/IEC JTC1/SC29/WG11 Document m12077, Busan, Korea, April 2005

- The G3D Graphics Engine: Advanced language features for simplicity and safety in a graphics API. *C/C++ Users Journal*, December 2004

- A Game Developer's Review of SIGGRAPH 2004. Gamasutra, Aug. 2004, http://www.gamasutra.com/features/20040830/mcguire_01.shtml *Reprinted by ACM SIGGRAPH Vancouver*, Oct. 2004, http://vancouver.siggraph.org/12_archive_2004-08.html

- M. McGuire, J. F. Hughes, K. Egan, M. Kilgard, and C. Everitt. Fast, practical and robust shadows. NVIDIA Tech Report, Austin, TX, Nov., 2003

- A Game Developer's Review of SIGGRAPH 2003. flipcode.com, July 2003, http://www.flipcode.com/misc/siggraph2003.shtml

- Five Ways I Improved My Coding Workflow. flipcode.com, October 10 2002, http://www.flipcode.com/articles/article_codingworkflow.shtml

- A Game Developer's Review of SIGGRAPH 2002: San Antonio. flipcode.com, July 2002, http://www.flipcode.com/misc/siggraph2002.shtml

- A Game Developer's Perspective of SIGGRAPH 2001. flipcode.com, July 2001, http://www.flipcode.com/misc/siggraph2001.shtml

- A Game Developer's Review of SIGGRAPH 2000: New Orleans. flipcode.com, July 2000, http://www.flipcode.com/misc/siggraph2000.shtml

Acknowledgements

I would like to thank the many people who contributed their experience towards the design of the SAMPL camera and algorithms, many of whom are also coauthors on my papers: Hanspter Pfister, Fredo Durand, Shree Nayar, Wojciech Matusik, Bill Freeman, John Barnwell, Bill Yerazunis, Doug Roble, and Michael Black. This project was made possible by the support of Joe Marks, the staff at MERL, and hardware and a fellowship from NVIDIA Corporation. The presentation of the matting work was greatly improved by comments from the anonymous SIGGRAPH reviewers.

My advisor and friend John “Spike” Hughes is a role model as a scientist and an educator. He has consistently gone the extra mile for me, and his hand is felt throughout my graduate work.

My wife Sarah illuminates my days. She also has edited my thesis drafts and has carried scientific apparati around Cambridge in the winter. (I would thank John and Sarah further, but that would only increase their editing task.) Joining Sarah in the snow was my friend and baked-goods provider, Jess Davenport.

I am grateful for the advice and support of many people at the Brown University Computer Science department who gave me a home and enthusiastic support for four years. These are in particular Andy van Dam, David Laidlaw, Tomer Moscovich, Liz Marai, Dan Keefe, Peter Sibley, Stefan Roth, Loring Holden, Mark Oribello, Olga Karpenko, Steve Dollins, Joe Laviola, and Sascha Becker of the Graphics Group; faculty members Shriram Krisnamurthi and Chad Jenkins; the administrative staff, especially Lori Agresti and Fran Palazzo; and the undergraduates from Games Group, Kevin Egan, Andi Fein, Pawel Wrotek, Alex Rice, Gabriel Taubman, Seth Block, Hari Khalsa, and Nick Musurca. Finally, special thanks to Corey Taylor from the G3D development team for helping with bug fixes at all hours of the night.

Contents

⋆ Parts of this dissertation have previously appeared and been submitted for publication in [57], [56], [85], [55], [54] and other papers with co-authors Wojciech Matusik, Hanspeter Pfister, John F. Hughes, Frédo Durand, Shree Nayar, and Anthony Vetro.

List of Tables

List of Figures

Chapter 1

Introduction

This thesis in particular describes a new way of thinking about images of real scenes, and of rendering virtual scenes without meshes. A multi-parameter video allows us to capture and convey more visual information than traditional photograph or video conveys to the naked eye. Multi-parameter video streams can help capture and visualize hard-to-observe phenomena from the real world. These include fast-moving objects, details concealed in the shadows of a bright scene, surfaces with hyperspectral reflectance patterns, camouflaged targets. The applications and motivation for visualizing each of these are clear. Multi-parameter videos also help visualize worlds that never existed: the results are special effects for films.

A video is parameterized on the camera settings chosen at the time of capture, encoding a limited information about the incident light field. If a camera can capture more information at each pixel, applications can later edit the video to communicate that information to a viewer (visualization), alter its appearance to meet an artistic goal while maintaining photorealism (visual effects), or synthesize a plausible new image from a virtual camera with a novel parameter set (virtualization). This thesis addresses all three.

There is a large body of literature describing systems for virtualizing a single parameter of an imaging system. By *virtualizing* I mean synthesizing new images corresponding to a virtual camera given different actual images from a set of real cameras. Virtual cameras are one of the places where computer graphics and computer vision meet, since the analysis of the real images is a vision problem and the synthesis of a new image is computer graphics. Examples of such work include the

high dynamic range work that virtualizes exposure, image based rendering and lightfield rendering that virtualize the optical center, high speed videography that virtualizes temporal sampling, and super resolution that virtualizes spatial sampling.

This thesis generalizes previous work by introducing multi-parameter video. It proposes a new video capture system that virtualizes aspects of the imager less explored in computer graphics, like the lens parameters, and allows multiple parameters to be *simultaneously* virtualized. This allows, for example, capture of video that is simultaneously multi-modal *and* high speed.

This thesis develops algorithms to solve problems in computer graphics using the new capture system. Although in most cases these are motivated by, and have direct applications in visual effects for film production, they also have applications in medical and scientific imaging, military, and remote sensing. Take for example the matting problem of separating subject from background. It is similar to the processes of separating a tumor from background in an X-ray and separating a target from background for identification in reconnaissance video.

1.1 Multi-parameter Video

A continuous 2D monochrome image is a real-valued function that measures incident light intensity at an imager. Video introduces a time dimension, giving a function $I(x,y,t)$.

Before capturing conventional monochrome video, the camera operator first chooses a single set of tunable *parameters* like focal length and imager position. During capture, sensor produces a stream that samples $I(\cdot)$ with high resolution. This is video with a single intensity value per pixel. In other words, each pixel contains a sample from a *single* set of imaging system parameters, e.g. the camera's focus depth. These parameters are "baked in" to the video. An application cannot later change a parameter like focus depth without recapturing the video. Because the entire parameter set of the camera is fixed (except for space and time) at a single set of values, I call traditional video *single-parameter video*. The actual video is discretely sampled, I switch to bracket notation: $I[x,y,t]$. Figure 1.1a shows a frame of single-parameter video.

Introducing color is the first step towards sampling across not only space and time but across the parameter space of the imaging system. Spectral response is one parameter of an imaging system,

and a color video sensor coarsely samples across the space of potential spectral responses. A pixel in $I[x,y,t,\lambda]$ contains multiple intensity values per pixel, corresponding to different spectral response parameters. I call the captured stream a *multi-parameter video* (color video is the simplest multi-parameter stream; I'll discuss more sophisticated streams in a moment.) Figure 1.1b shows the addition of color as another axis in parameter space.

Multi-parameter video allows an application to retroactively choose some of the imaging parameters. In the familiar case of three wavelength per pixel, an application can edit the spectral response by interpolating between channels or filtering across the color dimension. Because the human visual system can interpret three wavelength samples, $I[...,\lambda]$ does not strike us as an extraordinarily rich data stream or in need of visualization. However, consider the case where the camera samples more than three wavelengths, capturing more information about the incident light field and thereby providing more editing capability. Multispectral video quickly eclipses the capabilities of the human visual system, impressing us that it has moved spectral response from capture-time to edit-time. The key concept in adding a coarsely sampled dimension for λ is that it *virtualizes* the spectral response of the camera. That is, an application can now produce images with a virtual filter at edit time by interpolating between the real samples.

The camera can virtualize other parameters. For example, a high dynamic range (HDR) image virtualizes the exposure time, capturing a richer data set than a single-exposure image. It is a multi-parameter image; just as a color image contains multiple wavelength samples per pixel, an HDR image contains multiple exposure samples per pixel. Unlike a three-wavelength color image, however, it cannot be directly viewed in a meaningful way by a human observer. To create a directly viewable image, an application can simulate a virtual exposure or tone map the HDR data for a visualization that conveys more information than a single realistic photograph. Figure 1.1c shows a high dynamic range multi-parameter image created by. The cathedral image shown is in fact a common test case from the HDR literature. I didn't invent HDR or capture this particular image; my goal is to generalize our notion of images with varying parameters.

There is no reason to limit the virtualization to only one or two parameters of video. This thesis

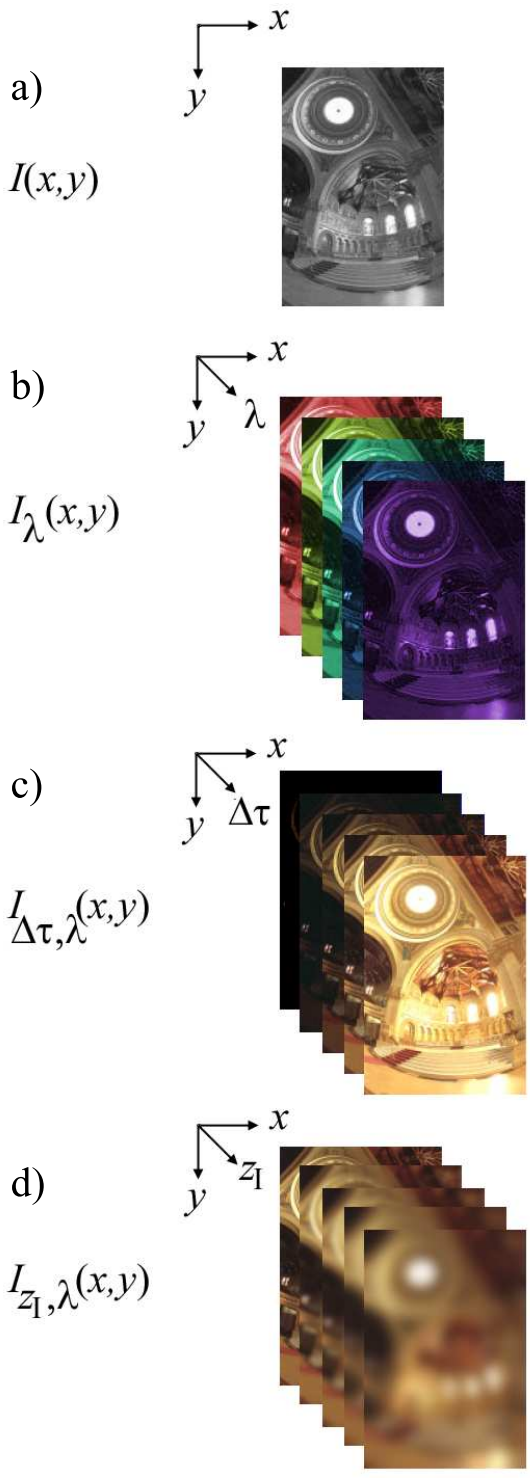


Figure 1.1: Comparison visualization of one- and two-camera parameter sampling spaces.

describes an imaging system called the *multi-parameter lens camera* that can be configured to virtualize focal length, lens-imager separation (i.e., "focus"), polarization response, spectral response, aperture radius, exposure time, center of projection, and spatial and temporal sampling. The pixels recorded by this system are truly multi-parameter and contain many samples that must be processed for meaningful viewing.

Figure 1.1d shows images that vary the distance between the imager and the aperture, creating varying amounts of defocus. Such multi-focus images are popular in computer vision literature.

This thesis describes explorations into part of the useful configuration space for the multi-parameter camera by developing new algorithms that take advantage of the rich data in multi-parameter video. Although the capture system *can* vary the center of projection (e.g., to provide stereo or light field sampling), for these algorithms I choose to hold the center of projection fixed, creating *single-center*, multi-parameter video. I chose to do so for three reasons.

The first is straightforward; I needed to explore how effective pure single-center video can be before expanding to multi-center. Second, changing the center of projection increases the significance of viewer-dependent effects and introduces parallax occlusion. In practice the camera I built is limited to 24 samples per pixel (up to three colors $\times$ eight video sensors), and in many cases the added information from multi-center images would be offset by the loss of data to parallax. However, I do identify situations in which multi-center video could could likely improve the results given a higher sensor count, and am applying the multi-sensor camera to other projects that can take advantage of its multi-center capabilities [53].

Third, the single-center system can be built compactly. Compared to a light field camera, a single-center camera has a significantly smaller footprint along two dimensions. If one forgoes the ability to change lens parameters, it is possible to split the view behind the objective and build a single-center camera of the same volume as a consumer video camera (e.g., 0.001 m^3). My camera that does support lens parameters is larger, with an optical path and components folded into a volume of about 0.025 m^3. However, in the near future it will likely be possible to build the same system with a much smaller footprint using either a nano-technology array of light-slowing spikes and microlenses or a holographic aperture.

1.2 Reader's Guide

This thesis spans a framework, a system, data, theory, and applications. The chapters are organized to build from first principles of the physics governing optics (at least, those relevant to the later discussion) up to the specific applications of interest to the film industry. However, the chapters are structured so that readers with a particular interest can begin directly with that area and only skim the other chapters to pick up cross references and notation. Each chapter also begins with an introduction accessible to the casual scientific reader. These introductions bring out the salient contributions of each chapter.

1.2.1 Framework

The multi-parameter framework described in chapters 2, 3, 4, and 6 builds heavily on previous work. These chapters include derivations that I have not previously seen expressed in a coherent end-to-end manner from light physics to capture devices, but the framework introduces no significant new mathematics and all equations will be familiar to readers acquainted with the previous work. The novel contributions of the multi-parameter framework are instead an idea: *different single-purpose capture devices all seek a common goal of increased sampling precision*; and two notation systems, one graphical and one symbolic, for compactly expressing the interesting ways that the precision is distributed among sampling parameters. This generalizes previous work in image capture and shows for the first time a way of directly comparing diverse capture systems like high dynamic range and high speed imaging systems.

Related work for the framework is referenced as the framework is presented. Related work for the system and applications is reviewed in chapter 5, after presentation of the framework. This allows the use of terminology and optical models introduced in the earlier chapters.

1.2.2 System

The hardware systems I developed for video capture is described in chapter 7.This include the crucial calibration processes and registration software. Appendices give relevant source code and

system diagrams, and most of the source code is available in electronic form on the Brown University web site. The data sets are too large to serve directly over today's internet but are available on request from Mitsubishi Electric Research Laboratory and Brown University.

1.2.3 Applications

The most detailed application of the SAMPL camera and multi-parameter theory is the centerpiece *defocus video matting* algorithm, which appears in chapter 8. This chapter describes new solution to the classic hard problem of matting. There was previously no solution for this problem that could handle unconstrained scenes without user assistance. I recommend reviewing the notation in chapter 4 and the matting related work in chapter 5 before beginning the matting chapter.

Chapter 9 demonstrates applications that span the breadth of the SAMPL camera's capabilities. It shows that the SAMPL camera can achieve high-quality results for different applications. The graphical notation for Optical Splitting Trees introduced in chapter 6 is used heavily in describing the algorithms.

Chapter 2

Models of Light

This chapter reviews physical properties of light, basic electromagnetic phenomena, and defines the terminology used to describe light. Readers concerned only with data capture and matting applications can skip this chapter.

Future chapters use a simplified ray model of light common in computer graphics. But before moving to that simple model it is important to first understand some of the complexity of electromagnetic phenomena. This guides intuition and makes clear where the simplified model is applicable. Because the small apertures used in this thesis approach the size where the ray model breaks down due to diffraction, and because sources of error like wavelength-specific variance and abberation cannot be expressed in the simple ray model, it is also critical to revisit physics to correctly evaluate potential sources of physical sampling error.

2.1 EM Waves

Light is an informal term for an electromagnetic (EM) wave with wavelength in the hundreds of nanometers, near the portion of the spectrum to which the human eye is sensitive. EM waves with longer wavelengths are classified as Microwaves and Radio. EM waves with shorter-than-visible wavelengths are classified as X-rays and Gamma rays.

An electromagnetic wave comprises an electric field $\mathsf{E}(x,t)$ and a magnetic field $\mathsf{B}(x,t)$ that vary sinusoidally in time (hence the "wave" designation) and are in phase with one another[1]. The

[1] Sans serif B and E are used exclusively in this chapter; in this chapter only, serif E represents energy in Joules. In

fields are vector quantities perpendicular to each other ($\mathsf{E}(x,t) \cdot \mathsf{B}(x,t) = 0$) and to the direction of propagation of the wave. The magnitude of the electric field is measured in units of Newtons per Coulomb or, equivalently, volts per meter, since it is defined as the negative spatial gradient of voltage (1 N / C = V / m = 1 kg m s^{-3} A^{-1}). The magnitude of the magnetic field is measured in Teslas (1 T = 1 kg s^{-2} A^{-1}). Note that, expanded into their SI base units, Teslas and Newtons per Coulomb differ by a factor of velocity units, m / s.

A *wave front* is the locus of points in space for which the magnetic fields and the electric fields all have the same phase. When visualized, wave fronts are typically rendered as a few subsequent crests of the electric field.

In the free space of vacuum, the wave fronts propagate[2] at the speed of light, $c = 299,792,458$ m/s $\approx 3 \times 10^8$ m/s. The speed of propagation is slower inside materials and can even become zero in some exceptional materials [7]. Slowing down light creates refraction, enabling the creation of lenses, and can be used to selectively shorten an optical path as proposed in the introduction. The magnitude and direction of propagation of energy flow due to a light wave is measured with the *Poynting vector*,

$$\mathsf{S} = \frac{1}{\mu_0} \mathsf{E} \times \mathsf{B}, \tag{2.1}$$

where $\mu_0 = 4\pi \times 10^{-7}$kg m s^{-1} A^{-2} is the *permeability of free space*. The units on the Poynting vector are kg s^{-4}.

The sinusoidally varying magnitudes of the electric and magnetic fields are related by $|\mathsf{E}(x,t)| = c|\mathsf{B}(x,t)|$ (the units check since the field units were related by velocity units).

Figure 2.1 shows the vector fields for a *linearly polarized* EM wave. The entire pattern moves through space at the speed of light.

The axis along which E is directed at point x is called the *polarization axis* of the wave at x (ignore the 180-degree reflection that occurs on the negative portion of the wave). The single wave shown in figure 2.1 is *linearly polarized*, that is, the axis of E is invariant in space and time. For polarization axis y, electric field with maximum magnitude E_{max} has the form:

---

the remainder of the thesis, $\vec{E}$ is an error vector and B is an image.

[2] However, it is possible to create a standing EM wave by placing a perfect reflector (ideal conductor) perpendicular to the direction of travel of a linearly polarized wave. The situation is analogous to fixing the end of a vibrating string.

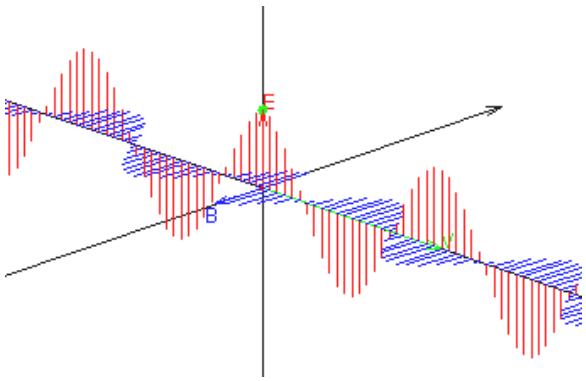


Figure 2.1: Electric and magnetic vector fields in a linearly polarized EM wave.

$$\mathsf{E}(x,t) = \mathsf{E}_{max} sin(\mathfrak{f}t + y \cdot x)y. \tag{2.2}$$

Now consider a scenario in which the fields of two in-phase linearly polarized EM waves of the same frequency superimpose. Let the polarization axes of the waves differ by angle θ. The net electric field is linearly polarized with peak magnitude $cos(\theta)|\mathsf{E}_1||\mathsf{E}_2|$. If the waves are instead out of phase by ϕ, the net electric field is *elliptically polarized*. In this case, the polarization axis at the wave front traces a flattened helix (spring shape) as it moves through space. When the phase difference is $\phi = \lambda/4$, the net wave is *circularly polarized* and E traces a perfect helix.

2.2 Photons

All types of EM waves are created by accelerating electrical charges. In the cases considered here, the charged particle that accelerates is an electron and it accelerates because it is either dropping from a higher energy orbital to a lower energy one or is accelerating under an electric field. Energy levels and electrons are discrete, so EM waves have a quantum nature– they can exist only at specific (albeit many) wavelengths and are produced in quantized packets. These packets are modeled as *photon* particles, which allows particle physics to model the interaction of light with surfaces. Although this model breaks down in some cases (e.g., diffraction grating) where the wave model is more accurate, it accurately describes situations where surfaces have features much larger than the wavelength of the EM waves considered.

When an electron drops from energy level E_2 to level E_1 (measured in Joules, 1 J = 1 N m = 1 kg m^2 s^{-2}), a photon is released with frequency

$$\mathfrak{f} = (E_2 - E_1)/\mathbf{h}, \tag{2.3}$$

where $\mathbf{h} = 6.626 \times 10^{-34}$ J·s is Plank's constant. The equivalent wavelength is

$$\lambda = \mathbf{c}/\mathfrak{f}. \tag{2.4}$$

Because all energy lost when the electron dropped was converted into the photon, the photon's energy is

$$E = \mathfrak{f}\mathbf{h} = \frac{\mathbf{h}c}{\lambda}. \tag{2.5}$$

There are several potential energy levels in each atom for an electron to begin and end the drop at. A material composed of a single element can therefore emit photons at many discrete wavelengths. The proportion of each wavelength present depends on the average temperature. *Black-body radiation* describes the expected distribution based on the temperature of an assumed heterogeneous material colored black so that reflected illumination and self-absorption can be ignored.

The visible spectrum ranges from about 400 to 700 $\times 10^{-9}$ m. The average wavelength is 550 $\times 10^{-9}$ m, and by equation 2.5, each photon at that wavelength carries energy

$$\begin{aligned} E_{visible} &\approx \frac{\mathbf{h}c}{\lambda}\Big|_{\lambda=550\text{nm}} \\ &\approx \frac{3 \times 10^8 \text{m s}^{-1} \cdot 6.6 \times 10^{-34} \text{J s}}{550 \times 10^{-9} \text{m}} \\ &\approx 3.61 \times 10^{-19} \text{J}. \end{aligned} \tag{2.6}$$

Within the visible spectrum, "white" light has approximately uniform energy with respect to wavelength. This means that there are more "red" photons than "blue" photons in white light because the red photons each have less energy. This physics argument leads to a valuable high-level observation: one can expect slightly higher variance in a camera's blue pixels than green, and higher in green than in red.

It is happenstance that uniform energy appears white to the human visual system. The Sun's energy is shifted towards the red end of the spectrum, appearing yellow, but indirect sunlight (i.e., the sky color) contains energy primarily at the blue end because of Raliegh atmospheric scattering, so the net illumination on Earth has is uniform energy. This can be observed on a winter day: snow appears white (yellow + blue) and shadows appear blue. The human visual system evolved under these illumination conditions, so it is equally sensitive at all wavelengths. Had we evolved on Mars, where the prevailing illumination is red, the eye would likely be hypersensitive to blue light and perceive the color we call pink as if it were white.

2.3 Rays

Modeling a single photon is rarely significant in computer graphics. An image is created by millions of photons striking the image plane. Because photons move so much faster than everyday objects, it is common to consider only the steady state in which a regular stream of photons is coursing along a set of paths through the scene. These paths are composed of linear segments, where the points where segments meet are interactions with surfaces or locations within a medium like fog that scatters light.

A *light ray* describes a point on a path and the direction to the next point in a path. The direction is the Poynting vector. The energy transferred along a ray is constant (in steady state), that is, there is constant photon flux at every point along the ray. A *bundle* or *pencil* of rays pass through a common vertex, forming a double cone. One particularly interesting vertex is the aperture of a pinhole camera, which is explored in depth in the next chapter.

2.4 Properties

The source of illumination for a scene can influence the kind of algorithm applied to its image. Light is classified according to the properties of its rays.

Natural light is the name given to light with a broad a distribution of wavelengths, polarizations, phases, and directions. It describes the illumination of an everyday scene under sunlight, fluorescent or incandescent bulbs, etc. Note that "artificial" (i.e., man-made) sources produce natural light under

this definition. Natural light is challenging to work with in computer vision because nothing can be assumed about the lighting environment. The new algorithms in this thesis work with natural light, which allows them to work on the most general scenes. Polarization issues arise only in the context of sources of error. In describing related work I refer to other kinds of light. These are used to actively illuminate a scene in order to make the specific problem at hand easier.

Structured illumination has a broad distribution of polarizations and phases but carries a pattern in wavelengths and the rays are all within a pencil. Projectors produce structured illumination; the wavelengths are described by the colors in the projected image and the apex of the pencil is the light source.

All of the rays in *collimated light* are parallel. Direct sunlight is collimated because the source is extremely far from the Earth. Semiconductor laser light is the most common man-made source of collimated light. In *coherent light*, all rays are in phase and at the same wavelength. This is also true of laser light, and is an important property in the creation of holograms.

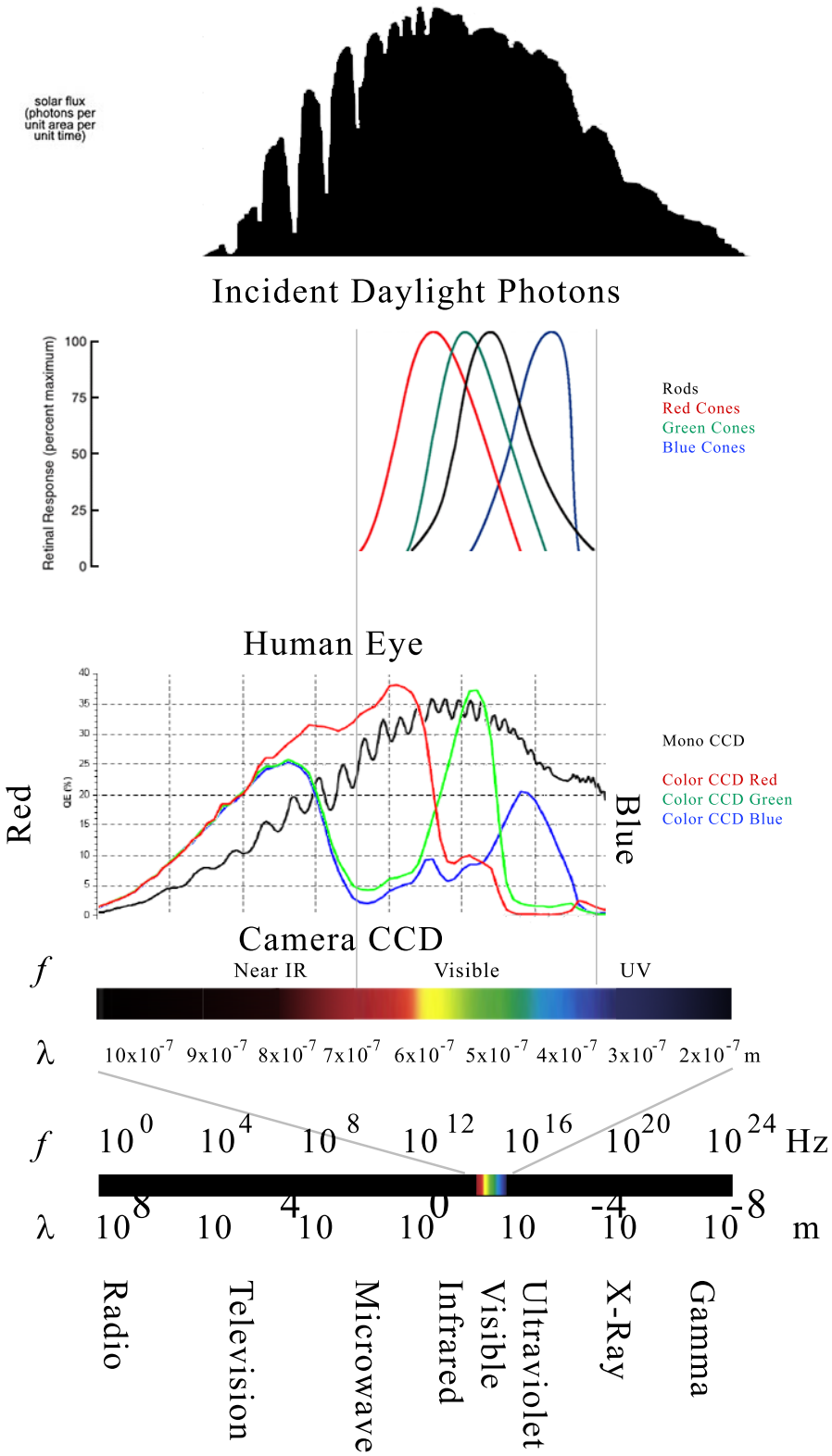


Figure 2.2: Daylight Spectrum and Human and CCD Spectral Responses

Chapter 3

Optical Image Formation

This chapter establishes basic camera notation and rephrases prior work so that new ideas can be expressed. It describes the interaction of light with an imaging system comprising a lens, aperture, and imager described by *parameters* like focal length and aperture radius. Most of the equations have been re-derived and motivated differently than in previous texts in order to address the agenda of multi-parameter videography and to present a consistent notation system across the optical model. The two-plane image and gradient models near the end of the chapter are especially popular topics in graphics and vision today; the derivation discussion here is intended to be intuitive and well motivated compared to the more in-depth work previously published.

The models presented in this chapter are the theoretical underpinning that allows the reasoning about elements of a 3D scene given multiple images with different parameters which appears in later chapters. These models were also used to synthesize images for experiments, which allowed exploration of both camera and algorithm design space before commitment to the design for a physical incarnation.

The equations used in this chapter are referenced throughout the thesis. Although many people have used cameras, fewer people are aware of how the parts of a camera control the image formed. The geometry behind the following facts is shown in this chapter and implicitly relied upon throughout the remainder of the thesis:

- Projected images are everywhere–but are too defocussed to recognize.

- An aperture (not a lens!) enables image formation.
- The lens determines the depth of field.
- The distance between aperture and imager determines image scale.
- Exposure time and aperture size determine intensity.
- At the plane of focus, a pinhole and lens image differ only in intensity.

3.1 Forward and Backward Models

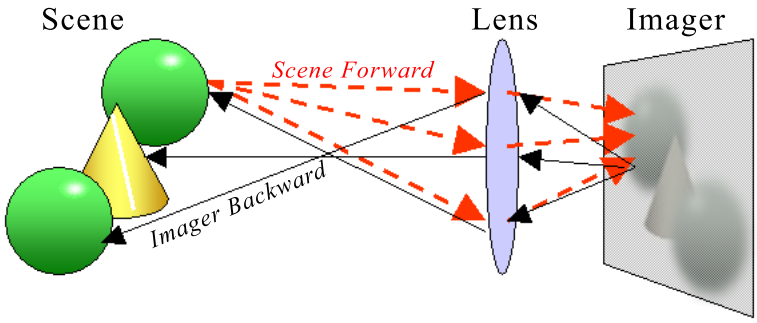


Figure 3.1: Schematic of imager-backward and scene-forward optical models.

The previous chapter began with a photon emitted during an electron's drop to a lower energy orbital and ended with light waves propagating through space along rays. I resume the discussion by treating light as a steady-state phenomena of transport rays using *ray optics*. I develop a number of relations for describing the interaction of light from a scene with an imaging system. These relations will progressively abstract the underlying physics until they reach *Fourier optics*, which models images as incident energy functions on the plane. I extend traditional Fourier optics to include occlusion and derive compact models of special cases of interest in computer graphics. These include scenes with distinct foreground and background elements that are to be distinguished by matting, and the pinhole camera frequently employed for rendering. Throughout I reference the notation and terminology of different fields including photography, optics, astronomy, computer vision, image processing, and computer graphics. I relate parameters from the mathematical model to measurable physical quantities in experiments with real sensors and scenes.

In computer graphics, image formation is often posed as an *imager-backward problem* for the integral of incident radiance over a pixel, where the scene is expressed as a 3D model of polygons and other primitives. Informally, this model pulls radiance from the scene to a pixel, as depicted by the black arrows in Figure 3.1. In the figure, the imager is on the right, the lens is in the center, and the scene is on the left. The imager-backward model follows paths from a pixel on the imager through the lens into the scene. These paths are shown as solid black arrows. These are the paths that a classic ray tracer follows. The scene-forward model instead distributes the contribution from each scene point across all pixels. The distribution paths are shown as red dashed arrows.

The scene-forward model of ray optics is a good framework for the rendering problem because the radiance at visible points is typically unknown and the same framework that traces rays backwards from the imager, through the lens, and to a surface can be further employed to follow the transport path back through the scene to a light source. In this work I am only concerned with radiance transport from a visible surface to the lens, and not the entire transport path. I therefore treat all points in the scene as emitters and assume some previous operation (e.g., a global illumination solver) has computed the appropriate amount of radiance to emit towards the lens. In this situation solving for the image arising from a scene, using good spatial subdivision, takes only linear time in the number of pixels.

In computer vision and image processing, image formation is often posed as a *scene-forward problem*. In this model, I solve for the radiance contribution to all pixels from a single patch in the scene. Informally, this model pushes radiance from a patch onto the imager, as depicted by the red arrows in Figure 3.1. It is convenient to represent the scene as planes of emitting points parallel to the image plane (for the optical axis-perpendicular imagers in most cameras, these are planes of constant depth). These planes are modeled as continuous functions over space, time, and wavelength, that is, videos. By holding time fixed, this becomes an infinite, continuous *image*, which is the primary building block of Fourier optics. The actual picture produced by a digital camera is a finite, discrete function produced by integrating the image over small patches of time, space, and wavelength.

When drawing distinctions between forward and backward models of image formation, the

pinhole camera is interesting because it resides between the models. Under a theoretically ideal pinhole camera, every visible point maps to exactly one point on the imager. I devote an entire section to pinhole optics because this property can be usefully exploited for both image synthesis and analysis.

3.2 Coordinate Systems

The equations of this thesis are expressed primarily in *lens space*, a right-handed 3D coordinate system where the optical axis is the *negative* z-axis, the origin is the center of the lens (i.e. the *optical center*), the y-axis increases vertically upwards, and the x-axis is given by their cross product. Because I work with a multi-sensor camera, there are potentially many lens spaces. However, the cameras are commonly configured so that their virtual centers of projection are identical.

Lens space is convenient for mathematical models of image formation where the lens remains fixed at the origin and the image plane moves during focussing. However, in a real scene, the image plane is usually fixed within the body of the camera and the lens moves. Thus it is common to measure distances into the scene relative to the image plane and then convert to lens space.

A 3D homogeneous space is used to express projections and linear transformations as 4×4 matrix multiplications. Two helper functions move points between an n-dimensional space (e.g., 3D lens space) and the corresponding homogeneous space with $(n+1)$ dimensions:

$$\text{inj}(x_1, x_2, ..., x_n) = (x_1, x_2, ..., x_n, 1) \tag{3.1}$$

$$\text{proj}(x_1, x_2, ..., x_{n+1}) = (x_1/x_{n+1}, x_2/x_{n+1}, ..., x_n/x_{n+1}) \tag{3.2}$$

Note that $\text{proj}(\text{inj}(p)) = p$ and $w\text{inj}(\text{proj}(x,y,z,w)) = (x,y,z)$ for $w \neq 0$. Injection and matrix multiplication are linear (and differentiable), while projection is nonlinear with a derivative that varies with w.

To drop dimensions (parallel projection), multiply by the generalized identity matrix with r rows and c columns:

$$ {}^{r}\mathbb{I}_c = \begin{bmatrix} 1 & & \\ & 1 & \\ & & \ddots \end{bmatrix} \begin{matrix} \uparrow \\ r \\ \downarrow \end{matrix} \\ \begin{matrix} \leftarrow & c & \rightarrow \end{matrix} \tag{3.3} $$

for example, ${}^{2}\mathbb{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$.

Pixel space (alternatively known as image space, film space, and screen space) is a traditional 2D raster coordinate system with origin at the upper left in which the *x*-axis increases to the right and the *y*-axis increases downward (relative to the content of the image– i.e. the ground appears at large *y* values and the sky at small values.)

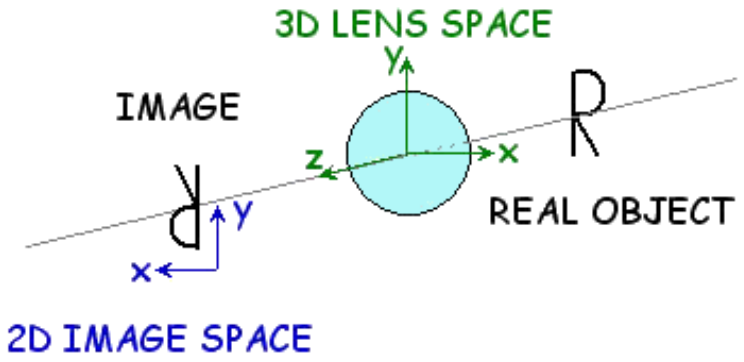


Figure 3.2: Image and lens coordinate frames

The equations for moving points between continuous pixel space (i, j) and 3D lens space *at the image plane* $q_z = z_I$, are:

$$ \text{fromPix}(i,j) = \left(W_I \left[\frac{1}{2} - \frac{i}{W_{pix}} \right], -H_I \left[\frac{1}{2} - \frac{j}{H_{pix}} \right], z_I \right) \tag{3.4} $$

$$ \text{toPix}(q) = \left(W_{pix} \left[\frac{1}{2} - \frac{q_x}{W_I} \right], H_{pix} \left[\frac{1}{2} + \frac{q_y}{H_I} \right] \right) \tag{3.5} $$

where W_I and H_I are the real-world dimensions and location of the imager. W_{pix} and H_{pix} are the pixel dimensions of the imager. The relation is invertible: $\text{toPix}(\text{fromPix}(i,j)) = (i,j)$.

Note that the pixel space coordinate system is rotated 180-degrees as mapped by toPix. This is because computer graphics typically works with positions on the *virtual image*, as if the image plane were in front of the lens. This is intuitive because the points project directly outward onto the scene with no flip, as if they were an image on a projector. It is occasionally useful to compute the mapping to the actual sensor pixel position, or in the terminology of optics, a location in the *real image*. Since the aperture flips images both vertically and horizontally, both axes are inverted on the real image compared to the virtual image. For reference, this mapping to the real image is given by:

$$\text{toRealPix}(p) = \left(W_{pix} \left[\frac{1}{2} + \frac{q_x}{W_I} \right], H_{pix} \left[\frac{1}{2} - \frac{q_y}{H_I} \right] \right) \tag{3.6}$$

The toPix mapping is useful for implementing end-to-end systems. Consider a camera and monitor placed back to back, with the video feed displayed live on the monitor. In this situation, toPix gives the screen coordinates that will produce the expected image, as if the monitor were a window.

By convention, the point at (i, j) in image coordinates is within integer pixel $(\lfloor i \rfloor, \lfloor j \rfloor)$.

It is useful to express toPix as a matrix product so that it can be chained to the end of a series of transformations in lens space. Let $q = \text{proj}(x, y, z, z/z_I)$. The $\text{toPix}(q)$ expression is then the matrix product

$$^2\mathbb{I}_3 \text{proj}(V[q_x, q_y, q_z, q_z/z_I]^{\text{T}}), \tag{3.7}$$

where V is the *viewport matrix*:

$$V = \begin{bmatrix} -\frac{W_{pix}}{W_I} & 0 & \frac{W_{pix}}{2z_I} & 0 \\ 0 & \frac{H_{pix}}{H_I} & \frac{H_{pix}}{2z_I} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The factors that translate the origin to the upper left of the screen are in the third column instead of the fourth, where one might expect translation terms, because the input was specified in a strange form with q_z/z_I in the w component. This form of input is intentionally chosen to match the output of the projection matrix introduced later in this chapter, which typically precedes the viewport matrix

in expressions.

#	f-number denominator (e.g., 1.4)
α	Area of pixel covered by the image of an object
a	Radius of the aperture
B	Image of a background plane
D_q	Depth of focus
D_p	Depth of field
d	Midfield depth *from the image plane*
f	Focal length of the lens
$\mathfrak{f}$	Frequency of a light wave
F	Image of a foreground plane
h	Point spread function
I	An image; intensity function at the image plane
∇	Gradient operator; $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$
p	A point in the scene ($p_z < 0$)
P	Perspective projection matrix
q	Location at which p is in focus ($q_z > 0$)
r	Radius of the circle of confusion
s	Width of a pixel $= W_I/W_{pix}$
$\Delta\tau_e$	Exposure time (shutter speed)
V	Viewport matrix
W_I, H_I	Real-world dimensions of the imager
W_{pix}, H_{pix}	Pixel dimensions of the imager
(x,y) or (i,j)	Image (pixel) coordinates
z_I	z-location of the imager/image plane (positive)
z_a	z-location of the aperture(negative)
$\otimes$	Convolution operator

Table 3.1: Symbols used in the lens model.

3.3 Lens Camera Model

I use a common parameterization of the optical system and scene in both image formation models. Assume an optical system consisting of a single lens, an image plane (containing the CCD array) and an aperture between them as shown in Figure 3.3. In the figure, the dark black labels refer to camera parameters. Light orange labels refer to properties of point p on the orange object at the upper right. Note that the z-axis increases to the *left*, so all points in the scene have negative z and

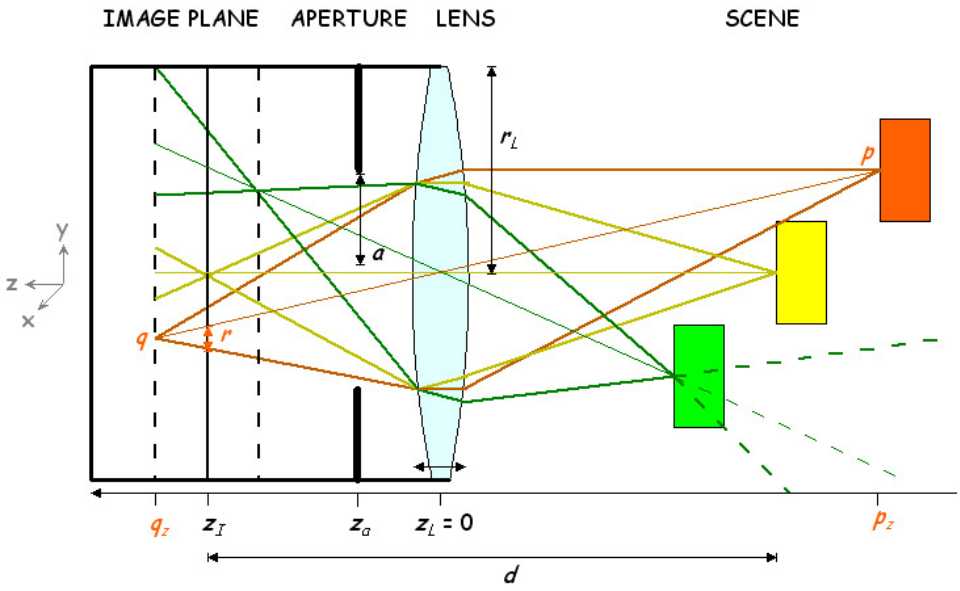


Figure 3.3: Ray model of a lens camera.

points behind the lens have positive z. The remainder of this section derives the geometry of the rays in terms of the camera parameters and location of point p.

3.3.1 Gaussian Optics

I use *Gaussian Optics* to approximate refraction at the lens. Gaussian (also known as *First Order*) Optics is predicated on the assumption that the angle θ between the optical axis and the vector to p is sufficiently small that trigonometric functions can be reasonably approximated by their first order Taylor terms, $\cos\theta \approx 1$ and $\sin\theta \approx \theta$. When the first order assumption is violated, the lens creates a defocussing effect known as *spherical abberation* that would not be present for an aspherical lens. For full derivations and discussion of the aspherical case, see Hecht's *Optics* text [42].

The locations of p and q are geometrically related by the lens shape and indices of refraction for the lens and surrounding medium by:

$$\frac{1}{-p_z - D/2} + \frac{1}{q_z - D/2} = \frac{\eta - \eta_m}{\eta_m}\left(\frac{1}{R_1} - \frac{1}{R_2}\right) + \frac{\eta}{\eta_m} \cdot \frac{D}{R_1 R_2} \tag{3.9}$$

where η is the index of refraction for the lens, η_m is the index of refraction for the medium, R_1 and R_2 are the radii of the spheres defining the front and back surface curves, D is the *thickness* of the

lens (the distance between the lens surfaces along the axis between the sphere centers).

For most imaging lenses, D is negligible compared to the radii of the spheres, i.e. the lens is *thin*. Assume that the medium is air ($\eta_{air} \approx 1.0$) and the lens is thin (the limit as $D \to 0$). The right side of the previous equation can then be approximated as the inverse of the focal length:

$$\frac{1}{f} = (\eta - 1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right). \tag{3.10}$$

This relation is known as the *Lensmaker's Formula*. Driving the D terms on the left side of equation 3.9 to zero and combining with the Lensmaker's Formula gives the *Gaussian Lens Formula*:

$$\frac{1}{f} = \frac{1}{-p_z} + \frac{1}{q_z} \tag{3.11}$$

which gives relative expressions for p and q:

$$p = \frac{qf}{f - q_z} \tag{3.12}$$

$$q = \frac{pf}{f + p_z} \tag{3.13}$$

The limit of equation 3.13 as $p_z \to \infty$ implies that objects at infinity are in perfect focus when the image plane is located one focal length behind the lens $z_I = q_z = f$.

Often the lens is replaced with a compound lens system to limit chromatic aberration. A compound lens can be modeled as a single lens with *power* $(1/f)$ equal to the sum of the powers of the individual lenses:

$$\frac{1}{f} = \sum \frac{1}{f_i} \tag{3.14}$$

Light paths from the scene that are incident on the lens at the origin are not bent by refraction and pass through the lens in a straight line. All other paths are refracted and form the two nappes of a double cone with vertex at q. If p is unoccluded from all points on the lens, these paths form an

image that is a uniform disk called the *circle of confusion*. Photographers also use the phrase "circle of confusion," however they typically refer to the *diameter* of the disk and not the radius.

When the aperture is immediately adjacent to the lens, i.e., $z_a = 0$, it effectively limits the radius of the lens and it is possible to compute the radius of the circle of confusion from the geometry in figure 3.4.

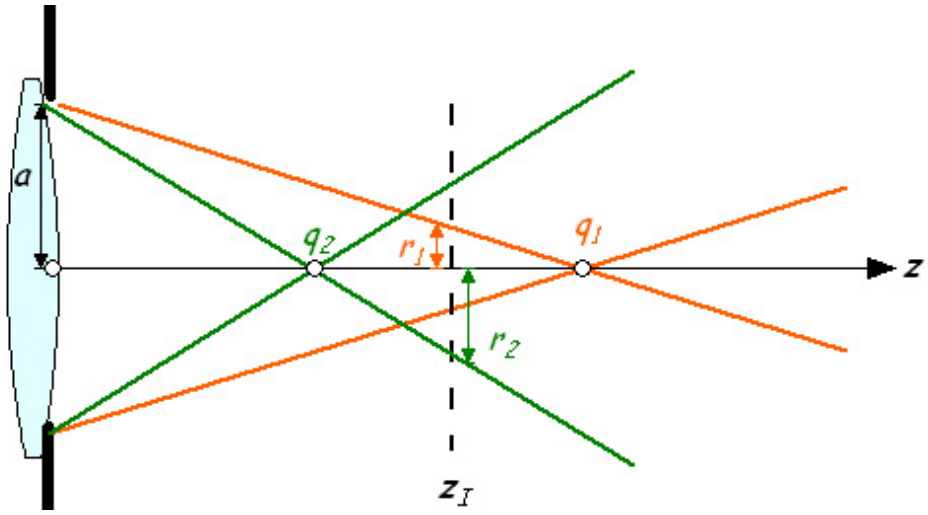


Figure 3.4: Paths from points at different depths converging behind the lens.

In the figure, points p_1 and p_2 (off the left, not shown) give rise to the light paths that focus at q_1 and q_2 behind the lens. Neither q is at z_I, so the circle of confusion has a non-zero radius. Similar triangles give the radius as a function of the cone apex:

$$\frac{r}{|z_I - q_z|} = \frac{a}{q_z} \tag{3.15}$$

$$r = a\left|\frac{z_I}{q_z} - 1\right|. \tag{3.16}$$

The sign of $z_I - q_z$ indicates whether the circle is flipped because the intersection is past the apex of the cone. This does not matter for a disk, but I revisit it later in cases where a full image is projected along the cone so that the intersection is no longer a disk.

Substituting for q_z by equation 3.13 gives the real-world image radius as an expression for the radius of the disk that is the image of a point p:

$$r = a\left|z_I \frac{p_z + f}{p_z f} - 1\right|. \tag{3.17}$$

Let the *imager* (e.g., CCD array or film) be a $W_I \times H_I$ rectangle able to resolve square elements (e.g., pixels) of size $s \times s$. Even when $r > 0$, a point will still appear reasonably *in focus* if its circle of confusion is below the resolution of the imager. The *field* is the interval over which p_z can be moved while maintaining this constraint; the *depth of field* is the span of the interval and the *depth of focus* is the span of the corresponding interval for q_z. Assume $a \gg s$, so that the circle of confusion is meaningful. Solving $\left(\frac{s}{2}\right)^2 = a^2\left(\frac{z_I}{q_z} - 1\right)^2$ for q_z gives the depth of focus:

$$\text{depth of focus} \quad D_q = \frac{4asz_I}{2a^2 - s^2} \tag{3.18}$$

$$\text{focus bounds} = \frac{a}{s}D_q \pm \frac{1}{2}D_q \tag{3.19}$$

Substitute the focus bounds into the Gaussian lens formula (equation 3.12) to solve for the field bounds and depth of field.

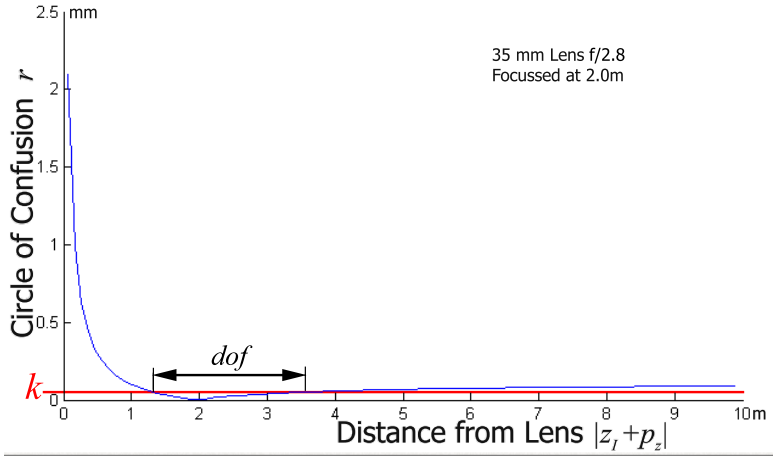


Figure 3.5: The circle of confusion grows rapidly as objects move closer than d, so the depth of field interval is shifted away from the lens.

Figure 3.3.1 shows a plot of r as a function of distance from the lens for a system focussed at 2m. The curve is hyperbolic so the circle of confusion grows much more rapidly as objects approach the lens than as they recede towards infinity. The red line shows $r = \frac{s}{2}$, the in-focus criterion, and its intersection with the curve marks the bounds.

Asymmetric lenses give rise to *astigmatism*, the imaging of a point off the axis as two perpendicular lines in different planes. The first convergence from the lens is the *primary image*, the second is the *secondary image*. There is no single point of convergence. In this case, the circle of least confusion lies along a curve called the *curvature of field*[94]. I assume symmetric lenses and neglect astigmatism.

I define the *field-of-view* of a camera as the angular dimensions that can be imaged with a very small aperture:

$$fov = 2tan^{-1}\left(\frac{W_I}{2z_I}\right) \times 2tan^{-1}\left(\frac{H_I}{2z_I}\right) \tag{3.20}$$

3.4 Pinhole Camera

Computer graphics often uses an *ideal pinhole camera model*. This is the limit of the lens camera model as $a \to 0$. Because it is linear in aperture radius, the radius circle of confusion is negligible when the aperture is negligible and the image of a point is a point even when $q_z \neq z_I$. Thus, the focal length and the presence of the lens are both irrelevant. Photographers sometimes say that the ideal pinhole "has an infinite focal length." This is not mathematically true, however it is true that the depth of field is infinite for an ideal pinhole.

One can construct a physical pinhole camera using a very small aperture and no lens. The practical aperture is limited to about $a = 0.25mm$ because smaller apertures produce diffraction effects that create a defocussed image, defeating the purpose of using a pinhole [42].

An important consideration for pinhole cameras is that the infinitesimal aperture requires either extremely bright lights or an extremely long exposure time to produce a useful image. This makes it impractical to produce pinhole images of dynamic scenes or to capture pinhole video.

Image formation is trivial under the ideal pinhole camera model. A point p in the scene projects to a single point $q = p\frac{z_I}{p_z}$ on the image plane. This mapping can be expressed with a homogeneous *perspective projection matrix*:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/z_I & 0 \end{bmatrix}, let q = \text{proj}(P \cdot \text{inj}(p)^T \tag{3.21}$$

Recall that $p_z < 0$ and $z_I > 0$. After projection $q_z = z_I$, that is, q is on the image plane.

3.5 Telephoto Lens

The only difference between the pinhole camera and the lens camera is the size of the circle of confusion, and not the location of projection. The lens radius and focal length do not appear in the projection equations; the projection matrix is independent of the focal length and aperture radius.

This may at first seem contrary to one's experience with telephoto lenses. A telephoto lens appears to flatten perspective and magnify the object viewed. However, consider the length of the barrel on which a telephoto lens is mounted (technically, it is a *telephoto objective*, and the lens is only the glass at the end of the barrel). The lens is very far from the image plane compared to a normal lens. Moving the image plane far from the lens always enlarges the image because the imager then intersects a larger cone of rays; a pinhole camera could achieve the same magnification effect. Because the image is larger more of it is cropped by the edges of the imager, which is why perspective appears compressed. This effect is called *telephoto compression*. Cropping also reduces the amount of light that falls on the imager, so telephoto lenses must be used with longer focal lengths.

The seeming paradox of "a lens cannot magnify an image" and "a telephoto lens magnifies an image" arises from poorly choosing the reference frame to be that of the photographer and camera. The phenomena is intuitive considered in lens space. Increasing the barrel length of the objective moves the image plane (and camera, and photographer) far away while the scene and lens remain fixed. This is analogous to moving a projector away from a screen– the farther back it is pulled, the larger and dimmer the image produced.

Moving the image plane far from the lens also increases the size of the circle of confusion. Aside

from the barrel length, the geometric difference between a normal lens and a telephoto lens is that the telephoto lens has a very long focal length ($>$ 75mm). This counteracts the increasing distance from the image plane and allows a reasonably small circle of confusion despite the magnification.

3.6 Obtaining Parameters from Real Cameras

I now relate the mathematical lens camera parametrization to the notation used with real cameras in order to apply this theory to real data.

The housing containing the aperture and one or more lenses is called an *objective*. Physical objectives have known focal lengths, commonly printed in millimeters on the lens. Adjustable zoom objectives are typically labeled with the compound focal length in both meters and feet on the adjustment ring.

An objective is focussed by moving the lens relative to the image plane. The distance labels on the focus adjustment ring give the approximate scene locations that will be in focus at those settings, i.e., the label gives the value of p_z that produces $q_z = z_I$ at that lens position. The manufacturer computes those distances by assuming a value for z_I (probably slightly larger than the focal length). There is some error because the objective manufacturer does not know the distance from the objective mount to the actual image plane in the camera. In practice, the labels are too coarse to use for actual measurement. The distance d between the image plane and the plane within the scene that is in sharp focus must be explicitly measured for use in computation. Recall that distance is measured relative to the image plane, because the lens moves relative to the scene during focussing.

To recover the image plane location from d, let $p_z = -d - z_I$, $q_z = z_I$ and solve equation 3.13 for the smaller z_I root:

$$z_I = \frac{1}{2}\left[d - \sqrt{d^2 - 4fd}\right] \tag{3.22}$$

The aperture is adjusted by a ring labeled with *f-numbers*, e.g. f/2.8, that opens and closes an iris. The aperture radius a of the iris is one-half the focal length divided by the denominator of the f-number (hence the "f" notation). The marked f-numbers, called *f-stops*, are scaled and rounded powers of $\sqrt{2}$ (e.g. f/1.4, f/2, f/2.8, f/5.6) so that the aperture area doubles between stops. By

definition, f-stop f/# gives aperture radius:

$$a = \frac{1}{2} \cdot \frac{f}{\#} \tag{3.23}$$

A film camera lifts a shutter for $\Delta\tau_e$ seconds to expose the film. This is called the *shutter speed* or exposure time and is indicated with a knob. A digital camera uses a charge-coupled device (CCD) or complementary metal oxide semiconductor (CMOS) array to replace the film. This thesis uses CCD cameras, which depend on the photoelectric effect. A CCD releases electrons in response to incident light, thus measuring the charge produced effectively "counts incident photons." Such a camera has no physical shutter, it simply measures and releases the charge after the exposure time.

The total radiant energy Q is linear in the exposure time and aperture area for any camera:

$$Q \propto a^2 \Delta\tau_e \tag{3.24}$$

To make high-speed images that preserve image intensity at the expense of depth-of-field, photographers therefore increase the aperture while decreasing the exposure time. This is why photographers often refer to f-stops as if they were a measure of time (e.g., "the exposure on my pinhole camera is f/500").

Film is typically described by the width in millimeters and resolution in line pairs per millimeter (lp/mm). In this measure, each pair of lines is the equivalent of four pixels–two foreground pixels for the lines and two background pixels to separate them from each other and the next line pair. "35 mm" (135 format) film has an image frame of 36×24mm and is rated at 50 lp/mm, roughly equivalent to a 34-megapixel imager.

Digital camera specifications directly report the dimensions and pixel resolution of the imager, although the actual output is frequently a subrectangle of the total image captured.

After projection under equation 3.21, q is measured in meters on the image plane, not pixels. Recall the *viewport matrix* that maps points on the image plane to points in pixel space previously introduced in equation 3.8.

The 2D, pixel-space *pinhole projection* of lens space point p is given by ${}^2\mathbb{I}_3\text{proj}(V * P * p)$. The matrix product VP is the familiar projection matrix used in computer graphics [36] extended to

real-world units and taking the mirroring produced by a lens into account.

3.7 Fourier Optics

Combining equations 3.17, 3.22, and 3.24 gives the radius r_p of the image of point p from directly measurable distances and camera settings:

$$r_p = \frac{1}{-4p_z\#}\left|\left(d - \sqrt{d^2 - 4fd}\right)(p_z + f) - 2fp_z\right| \tag{3.25}$$

where $f/\#$ is the f-stop, d is the mid-field distance from the image plane, f is the focal length, and $z_I - p_z$ is the distance to p from the image plane. To obtain a pixel radius, must divide by the width of a pixel, s (assume square pixels). The pixel width is occasionally called the *pixel pitch*.

Let $h_p(x,y)$ be a uniform disk of radius r_p/s with $\int\int h_p dxdy = 1$, where x,y are measured in units of pixels. This is the *point spread function* for the image of point p due to defocus.

Recall that p' is the pixel-space pinhole projection of p. Let $\delta_p(i,j)$ be the impulse that is inf where $(i,j) = p'$ and zero elsewhere with $\int\int \delta_p(x,y)dxdy = 1$.

The image of a scene containing only p is $I_p(x,y) = [\delta_p(i,j) \otimes h_p(i,j)](x,y)$.

The image of a scene containing a set of points **that do not occlude** one another relative to the aperture is the sum of the images of the individual points. It follows that if I_0 is the in-focus image of a textured plane perpendicular to the optical axis, $I = I_0 \otimes h_0$ is the defocussed image.

Consider a scene containing only two textured planes perpendicular to the optical axis. The *foreground* plane intersects the optical axis at z_F and the background plane intersects it at z_G (recall that z values are relative to the lens and points in the scene have $z < 0$.) Let the background plane's texture have full coverage everywhere (i.e., no alpha channel) and the image of the background plane be $B(x,y)$ when it is in perfect focus or, equivalently, imaged under a pinhole camera. Let the foreground plane have image $F(x,y)$ with partial coverage given by $\alpha(x,y)$.

Choose a point b on the background plane. The bundle of rays from b to the camera aperture forms a cone with apex at b that intersects the foreground plane at a disk. By similar triangles, that disk has radius $a\frac{z_B - z_F}{z_B}$, where a is the aperture radius. The transport from b to the aperture along each ray is modulated by $1 - \alpha$ at the particular point of intersection.

So far, this discussion considered only scenes containing a single background, plane. These corresponds to a two-plane scene where α is zero everywhere. In those cases the image of point b on the image plane was a uniform disk. Here it will be the occlusion shadow of the disk foreground. In other words, the point spread function of b in the two-plane scene is a sub-disk of the α image, divided by the area of the disk (regardless of the actual values of α).

Figure 3.6 shows the two-plane scene in exaggerated 3D perspective with discrete α values at the foreground plane. Every point on a defocussed background plane contributes a disk to the final image, cut by occlusions at the foreground plane. The disk shape arises from the circular silhouette of the lens; an octagonal iris gives rise to an octagon instead.

To write the image expression for this point spread function, must convert the radius and center of the disk from real-world 3D length and position to pixel coordinates on the foreground image. By similar triangles, the imager at z_I with real-world width W_I projects to a rectangle with real-world width $-z_F\frac{W_I}{z_I}$ on the foreground plane. The pixel radius of the disk *on the foreground image* where the cone from b intersects the foreground plane is thus:

$$r = a\frac{(z_F - z_G)z_I}{z_B s z_F}. \tag{3.26}$$

where s is the size of an imager pixel. Note that r is also the radius of the circle of confusion for the virtual image of b at plane z_F.

Let $\text{disk}_r(x,y)$ be the continuous function that is $1/(\pi r^2)$ when $\sqrt{x^2+y^2} \leq r$ and 0 elsewhere. Modulating α by a disk of radius r and shifting coordinates gives the point spread function for B at a given pixel (x,y), but the result is scaled in the pixels of α, not the pixels of the final image. A dilation is needed for the net spread function to the scale of the image of *b on the image plane*. Equation 3.17 gives this radius in meters. Converted to pixels it is:

$$r_B = \frac{a}{s}\left|z_I\frac{z_B + f}{z_B f} - 1\right|. \tag{3.27}$$

When computing the dilation factor $s = r/r_B$, the aperture radius and pixel size cancel, giving:

$$\sigma = \frac{z_F - z_B}{z_B z_F\left(\frac{z_B+f}{z_B f} - \frac{1}{z_I}\right)} \tag{3.28}$$

It is now possible to write an expression for the point spread function for B at (x,y),

$$h_{B,i,j}(x,y) = (1-\alpha(\sigma x+i,\sigma y+j))\text{disk}_{r_B}(x,y). \tag{3.29}$$

When σ is negative the intersection is past the apex of the cone behind the lens. After this point, the disk of alpha is upside down and left-to-right inverted.

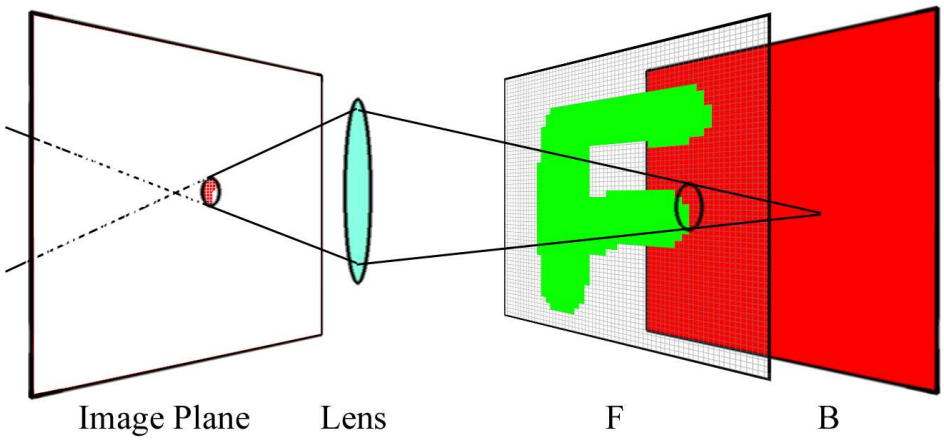


Figure 3.6: Two-plane scene with discrete α values at the foreground plane.

The composite image formed from the rays contributed by each plane is:

$$I = \int\int (\alpha F\,\delta(x-i,y-j)) \otimes h_{F,i,j}\;\partial i \partial j + \int\int (B\delta(x-i,y-j)) \otimes h_{B,i,j}\;\partial i \partial j$$

Note that at every (i,j) there is a different point-spread function and that α is hidden inside the h_B term.

Because F is never occluded, its point spread function is a disk with radius $r_F = \frac{a}{s}\left|z_I \frac{z_F+f}{z_F f} - 1\right|$. Subsituting this, the double integral over F is unnecessary and the expression for the image of two planes reduces to:

$$I = (\alpha F) \otimes \text{disk}_{r_F} + \int\int (B\delta(x-i,y-j)) \otimes h_{B,i,j}\;\partial i \partial j \tag{3.30}$$

One property evident from this equation is that the background appears in sharper focus near edges of the foreground because those edges narrow the effective aperture.

3.8 Special Cases

It is sometimes convenient to approximate equation 3.30 with the simpler form:

$$I \approx (\alpha F) \otimes \text{disk}_{r_F} + (1 - \alpha \otimes \text{disk}_{r_\alpha})(B \otimes \text{disk}_{r_B}), \tag{3.31}$$

where

$$r_\alpha \approx r_F. \tag{3.32}$$

This introduces error because it assumes that the point spread function for B has magnitude determined by α but uniform shape. The relation in equation 3.32 is also an approximation, although it introduces less error. I address the uniform approximation immediately and return to the radius approximation at the end of this section.

The error from assuming uniform point spread functions for B in equation 3.31 is small if either or both α and B are low frequency, or if at least one of the two planes is in perfect focus. There are three ways that at least one plane may be in focus: a pinhole image I_P, an image I_F focussed precisely on the foreground plane, and an image I_B focussed precisely on the background plane.

Figure 3.7 illustrates the three cases. It traces the cone of rays from a point on the foreground (solid green line) and a point on the background (dashed red line) through the lens and pinhole to various image planes. The point spread function h_F for the foreground in I_B, where the background is sharp, and point spread function h_B for the background in I_F, where the foreground is sharp, are shown as disks created by the intersection of the appropriate cones with the image planes. The specific equations for the three cases follow. I then bring all together and discuss their gradients, and then simplify further in cases where the frequency content of the scene is known.

3.8.1 Pinhole

If fs is very small or # is very large, then r is less than half a pixel at both planes. In this case, the image is the limit of the composite image as the aperture radius approaches zero. In equation 3.30, the point spread functions for F and B are (modulated) disks with radius proportional to a. These become (scaled) impulses in the limit:

$$I_P = \lim_{a \to 0} I = (\alpha F) \otimes \delta + \int\int (B\delta_{i,j}) \otimes ((1-\alpha)\delta_{i,j})\, \partial i \partial j \tag{3.33}$$

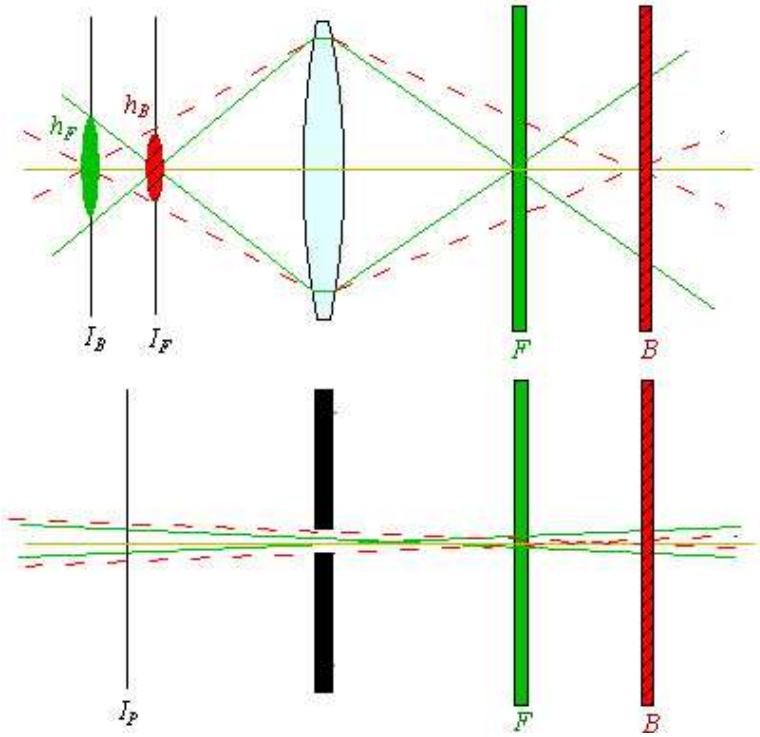


Figure 3.7: Schematic of cases where the image of two planes can be expressed simply.

Thus the composition simplifies to the well-known *pinhole composition* model first published by Porter and Duff [73], which is simple linear interpolation:

$$I_P = \alpha F + (1-\alpha)B\,. \tag{3.34}$$

The pinhole composition model is accurate when the PSFs involved are much smaller than the pixel size. It is an approximation when the PSFs are larger; if they are very much larger than a pixel it is an exceedingly poor model!

3.8.2 Focused on Background

When the background is in focus its PSF is an impulse (zero radius disk with finite integral). Rays in a cone from B are still modulated by a disk of $(1-\alpha)$ at the foreground plane, but that disk projects to a single point in the final image. Only the average value, and not the shape, of the α disk

intersected affects the final image. The composition equation in this case is thus:

$$\begin{aligned} I_B &= \lim_{\frac{1}{z_I} \to \frac{1}{f} + \frac{1}{z_B}} I \\ &= (\alpha F) \otimes \text{disk}_{r_F} + (B \otimes \delta)(1 - \alpha \otimes \text{disk}_{r_F}) \qquad (3.35) \\ &= (\alpha F) \otimes \text{disk}(r_F) + (1 - \alpha \otimes \text{disk}(r_F))B\,. \qquad (3.36) \end{aligned}$$

This model is accurate when the foreground is in focus. It is *not* an approximation; the math truly simplifies as described. Error is introduced only when the focus depth drifts from the foreground, the foreground object is large with respect to the depth of field, or when the background depths vary extremely with respect to the depth of field (recall that this section assumed planar foreground and background).

3.8.3 Focused on Foreground

When the background is defocused, its PSF varies along the border of the foreground object. The aperture shape is effectively different at each point because it is the intersection of the relevant disk of α and the true aperture. If the foreground is simultaneously in sharp focus, then the result is the limit of the composite image as the image plane moves to satisfy the Gaussian Lens Formula: $\frac{1}{z_I} \to \frac{1}{f} + \frac{1}{z_F}$. This causes $z_F f/(z_F + f)$ to approach z_I and the point spread function for F to again approach an impulse. Because the foreground is in focus, one can conceptually move the site at which background rays are modulated from the foreground plane (where the real occlusion occurs) to the image plane (which matches the resolution of α) where the modulation is easier to express. Under that model, the background contribution is the convolution of B with a disk, modulated at the image plane by $(1 - \alpha)$.

$$I_F = \lim_{\frac{1}{z_I} \to \frac{1}{f} + \frac{1}{z_F}} I$$

Of the three special cases, this is the one where equation 3.30 is complex and the equation 3.31 approximation is inaccurate. However, the approximation is so much simpler that it is necessary to make many algorithms (like the one discussed in chapter 8) practical for implementation. The PSF for the foreground drops out in this case and equation 3.31 becomes:

$$I_F \approx \alpha F + (1 - \alpha)(B \otimes \text{disk}(r_B))\,, \qquad (3.37)$$

The error in this approximation is at the borders of the foreground object. The smaller effective aperture at those locations should make the background contribution sharper (but dimmer) than where the background is completely unoccluded. The effect is similar to vignetting. The approximation ignores this effect, so the background appears more blurry than it should be. See Bhasin and Chaudhuri [12] and Asada et al. [5] for a more detailed discussion of the point spread functions at an occluding edge, improved (albeit more complex) approximations for the final case, and experimental results that validate the theory.

To recap, the collected reduced equations for the cases where one or more planes are in perfect focus are:

$$\begin{aligned} I_P &= \alpha F + (1-\alpha)B \\ I_F &= \alpha F + (1-\alpha)(B \otimes h_B) \\ I_B &\approx (\alpha F) \otimes h_F + (1 - \alpha \otimes h_F)B \end{aligned} \tag{3.38}$$

3.8.4 Gradients of Special Cases

Given these simplified equations, the spatial gradients can also be written in a simplified form. These are as follows (the extra terms emerge from the chain rule):

$$\begin{aligned} \nabla I_P &= (1-\alpha)\nabla B + \alpha \nabla F + (F-B)\nabla \alpha \\ \nabla I_F &= (1-\alpha)\nabla (B \otimes h_B) + \alpha \nabla F + (F - B \otimes h_B)\nabla \alpha \\ \nabla I_B &= (1 - \alpha \otimes h_F)\nabla B + \nabla(\alpha F \otimes h_F) + B\nabla(\alpha \otimes h_F), \end{aligned} \tag{3.39}$$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$. For a sufficiently narrow depth of field, the point spread functions are large disks. In that case, any term convolved with a non-impulse point spread function has a small gradient. Approximating such terms as zero, the second two equations simplify to:

$$\begin{aligned} \nabla I_F &\approx \alpha \nabla F + (F - B \otimes h_B)\nabla \alpha \\ \nabla I_B &\approx (1 - \alpha \otimes h_F)\nabla B \end{aligned} \tag{3.40}$$

When moving to discrete space for numerical computation, frequency aliasing and the fact that the gradients of blurry images are small but non-zero are sources of error in this approximation. For a real camera, imprecision in focussing, lens abberations, and a non-circular aperture are additional sources of error.

In continuous space, the special case equations in this section are limits of the general case, and not approximations.

So far, there have been no assumptions about the frequency content of either plane and the geometry has not been simplified in any way; the results stand in fully general circumstances. As a result, the equations are correct but unwieldy. Limiting the scene can dramatically simplify the equations. The next two sections make limiting assumptions about frequency content and approximating small gradients as zero to achieve such simplifications.

3.8.5 Low Frequency Background

When the background has no high frequencies, its gradient is small and can be approximated as zero. In this case, the image gradients are approximated by:

$$\begin{aligned} \nabla I_P &\approx \alpha \nabla F + (F - B)\nabla \alpha \\ \nabla I_F &\approx \alpha \nabla F + (F - B \otimes h_B)\nabla \alpha \\ \nabla I_B &\approx 0 \end{aligned} \tag{3.41}$$

3.8.6 Low Frequency Foreground and Background

When both background and premultiplied foreground have no high frequencies, their gradients are small and the image gradients are approximated[1] by:

[1] The low frequency ∇I_P expression was introduced by [60], including a discussion of point spread functions, and also appears in [84]

$$\nabla I_P \approx (F-B)\nabla\alpha \tag{3.42}$$

$$\nabla I_F \approx (F-B\otimes h_B)\nabla\alpha \tag{3.43}$$

$$\nabla I_B \approx 0 \tag{3.44}$$

3.8.7 Radius Approximation

I now return to the approximation $r_\alpha \approx r_F$ used in equation 3.31 and prove that the error in that approximation is small for typical scenes. Recall that r_α is the radius of the point-spread function applied to $(1-\alpha)$ during linear interpolation and that r_F is the point spread function applied to αF. Figure 3.8 depicts the geometry governing these variables.

The lens is in the center of the figure. The left side of the figure is 'in front of the lens' and the right side is 'behind the lens.' The green, thick, solid, vertical line on the far left is the background plane whose image in isolation is *B*. To its right is the foreground plane whose image in isolation is αF, represented by a red, thick, dotted, vertical line. The gap near the bottom of the foreground plane represents an area where $\alpha = 0$. Two vertical, colored lines on the right of the lens represent imager positions at which the background (green, solid) and foreground (red, dotted) are in perfect focus. The gray, dashed, horizontal line through the figure represents the optical axis.

Let W_B be the extent of the background plane that is within the frame. Let W_I be the extent of the imager that has W_{pix} pixels along that dimension. The ratio of these is the previously discussed pixel pitch *s* of the imager,

$$s = \frac{W_I}{W_{pix}}. \tag{3.45}$$

All variables are measured in a single spatial dimension, which here is *y*. The scene extent *W* imaged at location *z* (which is negative by my convention) in front of the lens is related to the size of the imager by similar triangles. These triangles are shown with gray dashed lines and give rise to the relation,

$$W \approx W_I \frac{z}{f}, \tag{3.46}$$

assuming $|z| >> f$. When that assumption fails, the object is near the focal distance and is nearly completely defocussed so that we cannot ignore vignetting effects.

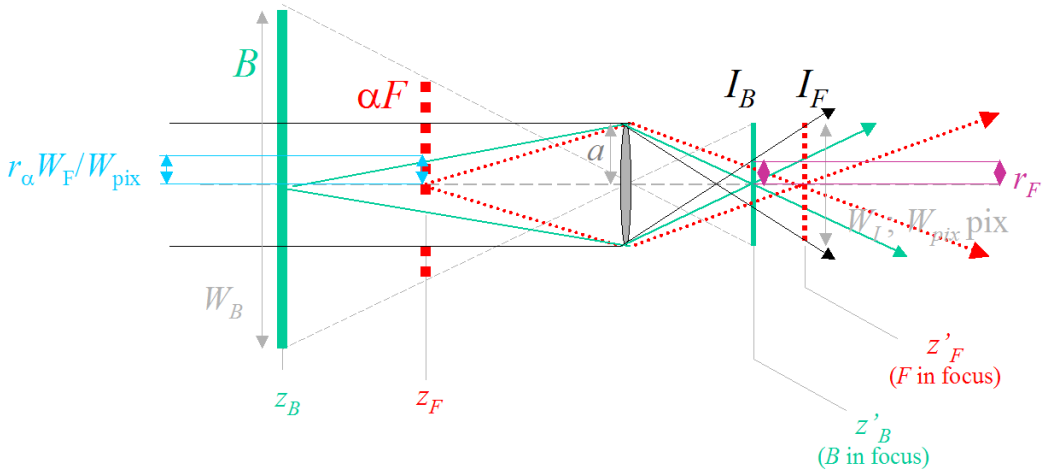


Figure 3.8: Geometry that determines point-spread radii r_F and r_α.

The two radii of interest are determined by intersections between cones and planes on opposite sides of the lens. Without loss of generality, consider the point on the foreground plane in front of the lens that happens to be on the optical axis. The boundaries of the pencil of rays from that point to the lens is shown by thin dotted lines. Behind the lens, they correspond to a second pencil with apex at depth z'_F, however we will not trace them all of the way to the apex. When the background plane is in focus, the foreground rays intersect the imager at depth z'_B and the intersection is a disk. The radius of that disk is r_F, which is here measured in pixels. It is shown in figure 3.8 as the right-most (purple) extent. Equations 3.17 and 3.27 gave expressions for computing the radius from different variables, derived by similar triangles. In terms of the variables labeled in the figure, r_F is given by:

$$\frac{r_F s}{z'_F - z'_B} = \frac{a}{z'_F} \tag{3.47}$$

$$r_F = \frac{a}{s}\left(1 - \frac{z_B(z_F + f)}{z_F(z_B + f)}\right) \tag{3.48}$$

Next, consider the point on the background plane in front of the lens (again, without loss of of generality) that is on the optical axis. The pencil of rays from this point, shown by thin solid green lines in the figure, converge behind the lens to a point on the imager. However, en route to the lens they intersect a cone of the foreground plane and are there modulated by $(1-\alpha)$. That intersection indirectly gives rise to r_α. The disk of intersection (in pixels) is the area of image α that modulates

a single point from B, however the dual of that statement is that each point in α affects an area of B of the same radius (in pixels).

The radius of the disk at the intersection of the foreground plane and the pencil of rays from a point on the background plane is marked by the left-most cyan extent in the figure. It gives rise to the relation:

$$\frac{r_\alpha(W_F/W_{pix})}{z_B - z_F} = \frac{a}{z_B}. \tag{3.49}$$

Solving this for r_α in pixels gives:

$$r_\alpha \approx \frac{af(z_F - z_B)}{sz_F z_B} \tag{3.50}$$

This is an approximation because equation 3.46 was used to remove the W terms. It therefore holds when $|z_F| >> f$. Note that the two radii computed are not exactly equal under equations 3.48 and 3.50. However, if one assumes that $|z_B| >> f$ and rewrites equation 3.48 as

$$r_F = \frac{a}{s}\frac{f(z_F - z_B)}{fz_F + z_F z_B}, \tag{3.51}$$

then the denominator is dominated by fz_F and the equation reduces equation 3.50. Thus,

$$r_F \approx r_\alpha \text{ when } |z_F| >> f \text{ and } |z_B| >> f, \tag{3.52}$$

which means that one can approximate the radii as equal when neither the scene nor the imager is near the focal length of the lens, where the image would be seriously blurred anyway. Thus the radius approximation contributes litter error in the approximate compositing equation 3.31 at the beginning of this section.

3.9 Transmission and Coverage

Thus far, I have only considered scenes where there is no transmission through the foreground and α represents partial coverage. In those scenes, the expressions αF and B are the energy emitted by the foreground and background in the direction of the lens. The foreground does not transmit light, so wherever the foreground emits non-zero energy, the background must be blocked. In continuous space, α is binary, so each image point is from either the foreground or the background exclusively. In discrete space, each pixel value is a sample of continuous function I_P convolved with a pixel

response function (e.g., a box filter for a camera CCD). This gives rise to the fractional α values described by Porter and Duff.

I now return to continuous space (binary α) and extend the model of the foreground object with four coefficients modeling its interaction with light. Let the foreground be a perfectly diffuse material with index of refraction matching the surrounding medium. Assume that, as do most real materials, it reacts identically to light from the front and behind.

Let $\hat{r}$, γ, and $\hat{a}$ be the fractions of incident light reflected, transmitted, and absorbed (the hats distinguish them from the radius and aperture variables used elsewhere in this chapter). For conservation of energy, at every frequency λ, $\hat{r}+\gamma+\hat{a}=1$. The material may also emit light with intensity E. Let L describe the total incident light contribution from the front (taking angle of incidence into account) and B describe light from behind. All variables may vary over x, y, and λ. The observed light intensity under pinhole projection is:

$$I_P = \alpha(E+\hat{r}L+\gamma B)+(1-\alpha)B \tag{3.53}$$

As before, the first term describes locations where the foreground covers the background and the second where the background is uncovered. The light absorbed by the foreground is $\alpha\hat{a}(B+L)$. I ignore the difference between emitted and reflected light and abstract them both into $F=E+\hat{r}L$. This gives:

$$\begin{aligned} I_P &= \alpha F+(1-\alpha(1-\gamma))B \\ \nabla I_P &= \alpha\nabla F+F\nabla\alpha+(1-\alpha(1-\gamma))\nabla B+\alpha B\nabla\gamma-(1-\gamma)B\nabla\alpha \end{aligned} \tag{3.54}$$

Many highly transmissive materials encountered in daily life, like colored glass, do not emit and have very low reflectance, so $F\approx 0$. The impression of the color of such a surface is given by the wavelength sensitive γ_λ function. For example, a transmissive red gel on a black backing appears nearly black, and the same gel with a white backlight[2] in a dark room appears red. Opaque

[2]This thought experiment employs a backlight instead of a white backing to eliminate the question of the double

red construction paper appears red regardless of the backing color. Neither gel nor paper emits light, so the difference lies in the wavelengths reflected, transmitted, and absorbed. The "red" paper *reflects* red light, absorbs other wavelengths, and transmits nothing. The "red" gel *transmits* red light, absorbs other wavelengths, and reflects nothing. The final possibility is an ideal beam-splitter, which absorbs nothing and partly transmits and reflects all wavelengths.

Although partial coverage and transmission are separate phenomena, they are frequently conflated in computer graphics for two related reasons. The first is that linear blending gives the visual impression of transparency. Consider the (discrete) case of a uniform foreground plane with $\alpha = 0.5$ and $\gamma = 0.5$. By equation 3.54, the resulting image has intensity $I_P = 0.5F + 0.75B$ (brighter than either foreground or background!), which is not a linear interpolation between foreground and background. So-called "colorless" glass is a special case where linear interpolation does suffice for transparency, as described in the next section. The second reason is screen transparency. From far away, a fine window screen with $\alpha = \frac{1}{2}, \gamma = 0$ is indistinguishable from colorless glass with $\alpha = 1, \gamma = \frac{1}{2}$ because each effectively absorbs 50% of the light from the background.

It is worth mentioning that the "one-way mirror" is not a case of transmission acting differently in different directions. Half-silvered mirrors are beam splitters that transmit about 50% of incident light and specularly reflect the remainder. When objects on one side of the glass are brightly illuminated and objects on the opposite side are dim, this gives the illusion of one-way visibility. Were both sides equally illuminated observers on opposite sides would see equally bright superimposed images of themselves and their counterparts.

3.10 Imager and Noise Model

The values returned from a real digital camera are discrete in space and value, have limited dynamic range, and are noisy.

I model the true image $I(x, y, \lambda, t)$ convolved with the pixel response function, $w(x, y, \lambda, t)$, and sampled at regular intervals before it hits the analog-to-digital reader. The sensors are manufactured

filtering. That occurs when light passes through the gel to the backing, reflects, and passes back through the gel again en route to the observer.

so that the response function is nearly a 4D box filter, which uniformly integrates incident illumination across an entire pixel, the entire visible spectrum, and exposure time. In practice the response must be zero at the narrow spatial border separating adjacent pixels and is not perfectly flat across the surface. There are also some other electronics for each pixel that are printed on the circuit beside the actual photosensitive area. The *fill factor* is the ratio of photosensitive area to total pixel area, which can be less than 50% for some CCD designs.

There is also a temporal border because the exposure period cannot exceed the inter-frame time and there is some downtime while the device discharges the CCD pixels to read their values. The wavelength response is not only non-uniform but also extends outside the visible range, particularly on the infrared side, making CCDs extremely heat sensitive. The CMOS response function is shown in figure 3.9 from [1].

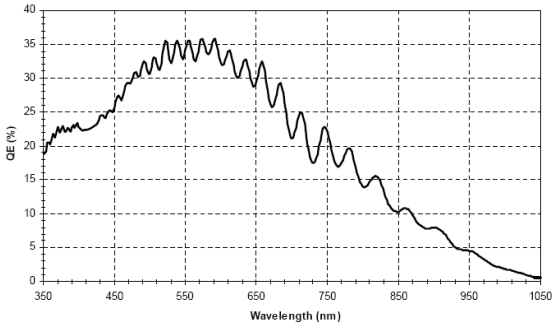


Figure 3.9: Spectral response of a CMOS image sensor. From Kodak.

The low-pass filtered image is converted from analog to digital at a fixed bit depth. Digital gain and brightness apply a linear transformation to the values and then the final result is rounded to the nearest integer and clamped to the typical 8-bit dynamic range:

$$I_{digital} = \max(0, \min(255, \lfloor \lfloor I \otimes w \rfloor * gain + brightness \rfloor)) \tag{3.55}$$

Color cameras have three pixel response functions w_r, w_g, and w_b, to sample across the spectrum. Most color cameras do not measure red, green, and blue values at each pixel but instead tile a color filter array (CFA) over pixels to give them different wavelength response functions. Bayer's popular tiling pattern [8] prescribes a regularly tiled grid. Even pixel rows alternate between green

and red pixels and odd pixel rows alternate between blue and green (figure 3.10. Twice the space is given to green because the human eye has the most resolution in green, which is also centered on the visual spectrum. The color filters typically do not match the three color response functions in the human eye, which leads to desaturation of the captured image. Figure 3.11 shows the spectral response of red, green, and blue pixels (from [1]).

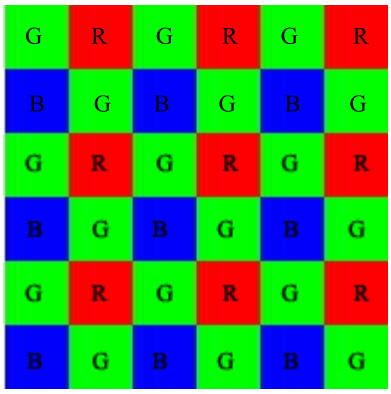


Figure 3.10: The Bayer pattern of CCD pixel tiling.

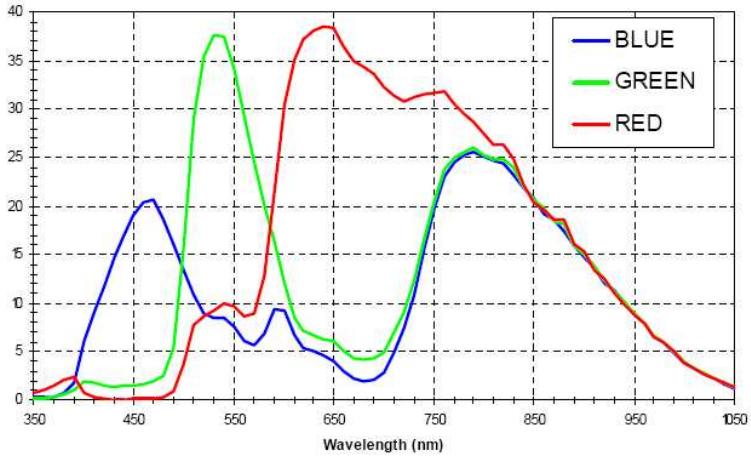


Figure 3.11: Product of a typical RGB CFA spectral response with that of a CMOS image sensor. From Kodak.

Moreover, the individual sensor pixels differ from the manufacturing target, which is itself more conservative than the theoretical ideal because of practical constraints like non-zero pixel borders and the difficulty of manufacturing perfect box filters [3].

[3]Regardless, the ideal filter is probably a sinc function and not a box, since the box produces ringing in the frequency

There are two common pixel errors that change slowly over time. *Hot* pixels return values that are too high and *cold* pixels return values that are too low. Because the CCD is sensitive well into the infrared range, these names are not just evocative but often actually descriptive. As the sensor heats up from operation or bright illumination in the scene more pixels become hot due to ambient IR energy. Although this error can also result from manufacturing imprecision, any slowly-changing bias is called a *thermal error* in the pixel value.

It is common practice to model the hot/cold error as a per-pixel linear function. The parameters of this model are obtained from two photographs. A *black level* (*dark field* in astronomy) image is captured with the lens cap on and a long exposure. This determines whether the pixel is hot and will return a positively biased value even in the absence of visible light. A *gray level* (*flat field* in astronomy) image is created by imaging a uniform gray card with a long exposure. The image of the card is defocussed by placing the card outside the field. This distributes the illumination from each point on the card approximately evenly across all sensor pixels. Defocussing compensates for non-uniform reflectance off the card due to uncontrollable surface properties (e.g., specularity, roughness, non-uniform albedo). The gray level alone can be used to identify cold pixels. Combining both images gives the model of manufacturing and thermal noise: $I = 2I_{true}I_{gray} + I_{black}$. The multiplication by two is a division (normalization) by the $\frac{1}{2}$, the expected value of I_{gray}. The value is exactly $\frac{1}{2}$ and is unitless because the algorithm operates on pixel values arbitrarily scaled to the range $[0,1]$ and not real energy units.

In addition to the thermal noise images, each frame has a small amount of sampling error that is called *transient noise* image noise. This primarily arises because light is a stream of photons and, for short exposures, the variance on the individual photon count at a pixel per frame is high. Because the electronics for the sensor are mounted to the same breadboard as the CCD, there is also some electrical interference that causes small frame-to-frame variation in the images returned for a static scene.

How many photons hit a CCD pixel per frame? Direct sunlight carries about 500 W/m^2 (1 W = 1 J/s) of energy in the visible spectrum. Because each visible photon carries about $E = 3.61 \times 10^{-19}$J

domain.

(equation 2.6), there are about 1.4×10^{21} photons incident per second under direct sunlight per square meter, per second. Consider a camera with a 50 mm lens, 640×480 imager, and wide open aperture directly imaging the (completely defocussed) sun with an exposure of $\Delta\tau = 1/30s$. The aperture has area $A = \pi \frac{.05\text{m}}{1.4}^2 \approx 0.004\text{m}^2$, so during one frame there are $\frac{1.38\times10^{21}0.004\text{m}^2}{30}$ photons directed at the imager. The imager contains about 3×10^5 pixels, so each pixel receives about 6×10^{11} photons per frame. This is an upper bound–the camera is usually directed at a surface, not the sun, there is some light loss through the lens, pixels are not fully covered by their photosensors, and CCDs have quantum efficiencies of about 0.5 (i.e., only half the incident energy is converted to measurable charge). When imaging a sun-lit scene one can therefore expect photon counts for bright areas to be on the order of 10^9, and indoors on the order of 10^5.

3.11 Sample Values

To give a sense of reasonable parameters and properties of a camera, below is a list of values for the 640×480 Basler a601f cameras with 35 mm Cosmicar lenses used in this thesis. These constants are for a typical scenario where the f-number is $f/1.4$ (wide open) and the objective is focussed 2m from the image plane.

$f = 3.5 \times 10^{-2}\text{m}$

$\# = 1.4$

$W_I \times H_I = (7.2 \times 5.4) \times 10^{-3}\text{m}$

$W_{pix} \times H_{pix} = 640 \times 480\text{pix}$

$d = 2.0\text{m}$ (from the lens)

$a = 1.17 \times 10^{-2}\text{m}$

$z_I = 3.56 \times 10^{-2}\text{m}$

$s = 1.562 \times 10^{-5}\text{m/pix}$

$fov = 16.0 \times 12.0^\circ = 0.278 \times 0.210\text{rad}$

$$VP = \begin{bmatrix} -106667 & 0 & 0.1684 & 0 \\ 0 & 106667 & 6735.0 & 0 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 28.062 & 0 \end{bmatrix}$$

Figure 3.4 is a graph of r vs d for this scenario.

Chapter 4

Multi-Parameter Notation

This chapter formally defines multi-parameter video and its notation based on the optical model from the previous chapter. Because Matlab is a natural tool for working with multi-parameter images, along with the definitions I give notes on how they interact with that implementation environment.

The idea of varying a single parameter (like focus) while gathering a collection of images has been used previously in graphics and vision. The contribution of this chapter is a generic framework for describing organized collections of multiple parameter values. The framework is suitable for the design of image-capture hardware and applications.

4.1 Functions on the Imager

In linear algebra, a matrix is most often used as a compact notation for the coefficients in a set of linear equations. It is common practice to use a matrix (as in 'a grid') of numbers to represent images as well, where the matrix indices describe the pixel at which a sample lies. This practice allows convenient re-use of the grid structure and notation from linear algebra. Pointwise operations like addition have identical implementations for the two structures, and the same efficient data formats (e.g., sparse vs. dense, floating point vs. integer) are available for each. Since linear algebra is an important tool in image processing, using a matrix to represent an image also has the benefit of allowing a smooth transition between imaging operations and linear algebra operations. Matlab and other numerics tools exploit all of these properties by allowing images and sets of

coefficients to be used interchangeably. The only limitation is that certain operations that make sense in linear algebra—row exchanges, for example—do not make sense in image processing, and must be avoided. When something like a row permutation *is* needed (e.g., when solving for an inverse transformation), the typical strategy is to unravel images into vectors of pixel values and indices, perform the math, and then pack them back up. This practice is described later in this chapter and used extensively in chapter 8.

A *multi-parameter video* is a discrete function on finite integer domains. It is frequently implemented as an *n*-dimensional array (which, in packages like Matlab, is aliased with the concept of "matrix"). To be clear, a multi-parameter video is *not* a matrix in the linear algebra sense. However, as is the common case for the Matlab user, one can read this thesis thinking of them as matrices and rarely become confused. The only point requiring care when reading is that two adjacent image variables denote componentwise or array multiplication ("$.*$" operator in Matlab), and not matrix multiplication. As with any componentwise operation, the operands must have matching shapes. This allows αF to denote the pixel-by-pixel modulation of video (or image, if time is held constant) F by video α, which is the notation expected in computer graphics.

The scalar values of a multi-parameter video represent the samples of a continuous function at specific locations in parameter space. The function being sampled is the integral of the continuous image formation equation at the image plane. Let $L(x,y,t,\omega)$ represent the incident radiance at the image plane at point (x,y) at time t from direction ω. Radiance is measured in W m^{-2} sr^{-}1. Integrating L over the hemisphere visible from (x,y) eliminates the steradian denominator and yields the continuous space image:

$$I(x,y,t) = \int_{\text{hemi}} L(x',y',t',\omega')\, d\omega'. \tag{4.1}$$

Integrating $I(\cdot)$ over a small space ($dx\ dy$, a pixel) and duration (dt, an exposure) gives energy units of Joules. This represents the energy absorbed by a discrete pixel in a photograph. That integration is what is sampled by the discrete multi-parameter video function:

$$I[x,y,t] = \int_{x-\Delta x/2}^{x+\Delta x/2} \int_{y-\Delta y/2}^{y+\Delta y/2} \int_{t-\Delta\ t/2}^{t+\Delta\ t/2} w(x'-x,y'-y,t'-t) I(x',y',t')\ dt'\ dy'\ dx', \qquad (4.2)$$

where $w(\cdot)$ is the imaging filter and its extents are Δx and Δy in space and Δt in time. (Note the change from parentheses to brackets for I). For a typical digital video camera, w is a box filter in both space and time, where the box has about the extent of a pixel in space and an exposure in time. Equation 4.2 describes a monochrome video; if t is held fixed it represents a single frame. Returning to the case where t varies through a discrete set of values, image I can also be viewed as a three-dimensional array. This is the monochrome version of the *video cube* representation that is now popular, particularly in non-photorealistic rendering [32, 48].

4.2 Parameters

Horizontal space, vertical space, and time are three of many interesting sampling *parameters*. The other parameters are less frequently discussed in the literature but could just as well appear explicitly on L as well. Any parameter of the imaging system may appear inside the brackets of I. Some interesting ones are:

- Wavelength (λ; "color")
- Imager depth (z_I)
- Sub-pixel space sampling offset
- Sub-frame time sampling offset ("time offset")
- Aperture
- Polarization phase
- Wave or "complex" phase

When defining the sampling process, the imaging filter w must be extended with these parameters as well. It will not always be a box filter. For example, in the case of λ, a typical CCD camera's filter is described by the plots in figures 3.9 and 3.11 from the previous chapter.

It is also acceptable to sample derived parameters that correspond to no particular part of the physical capture system but are instead derived values. For example, chapter 8 combines imager depth, aperture size, and a post-processing scale correction into a derived parameter called *z* that selects between a near-focussed sub-image, far-focussed image, and a pinhole image.

The specific parameterization chosen for a problem is thus the choice of the implementor; success at finding a solution will likely rest on whether the chosen parameterization is well-suited to the problem. Although it is useful to chose statistically independent (orthogonal) parameters, it is frequently not practical to do so. For example, the blue and green channel's filter responses overlap significantly for the typical CCD, so samples corresponding to different wavelengths have non-zero correlation.

4.3 Range, Domain, and Shorthand

As described, a multi-parameter video is a real-scalar-valued discrete function. It measures integrals of radiance, so the range is the set of non-negative numbers. In practice, the range is frequently linearly scaled to a unitless [0, 1] range where 1 is the brightest value in a video with no over-exposed pixels and 0 is the dimmest value.

What is the domain of a video? Because any parameter of the imaging system, whether fundamental or derived, is eligible to act as a parameter of the video, there is no fixed domain. Before working with a particular multi-parameter video I always describe its parametrization. Any fundamental parameter not in that parameterization is assumed to be held fixed for the experiment. Note that holding a parameter fixed does not imply a narrow filter for it; e.g., in the case of wavelength, a monochrome image has a single λ sample but the the extents of *w* are large for that parameter because all visible wavelengths are in the domain of integration.

Because there is a fixed parameterization for each experiment, the reader can detect missing parameters in an expression. Missing parameters are used as a convenience notation for sub-domains. For example, take the context of the space-and-time parameterization from equation 4.2. In that context, the value of an expression like $I[x,y]$ is a new discrete function of one parameter, *t*, i.e.,

it is the 1-pixel video over all time[1]. This concept is straightforward if one slips into thinking of multi-parameter videos as multidimensional arrays; $I[x,y]$ is a sub-array, which is denoted `I(x, y, :)` in Matlab. This notation means that in an equation it is safe to read omitted parameters as ones that do not vary in that equation. For example, when parameters λ and t are omitted in an equation (as they often are), that equation holds for all λ and t.

Many equations have the same form in x and y, but including both spatial dimensions creates burdensome notation. As is common in image processing literature, I frequently omit y and describe the equation as if the image were "1D." This contradicts the missing parameter notation, but I trust it will be obvious to the reader what is meant in each case.

4.4 Operators

All of the usual pointwise arithmetic operators are available to apply to two multi-parameter videos on the same domain. These are inn particular: addition ($a+b$), subtraction ($a-b$), multiplication ($a*b$ or $a\ b$), and division (a/b). Where one video's parameters are strict subset of the other's, let the video with the smaller parameter set be spread. For example, let modulation of a color video by a monochrome video produce a color video:

$$\alpha[x]F[x,\lambda]\big|_{x',\lambda'} = \alpha[x']F[x',\lambda']. \tag{4.3}$$

Let scalar constants be spread across all parameters. Note that subtraction may produce multi-parameter images with negative values ($1-\alpha$ is a common one).

Let the spatial convolution $F \otimes G$ of two videos have the same domain as F. It is computed by first extending the spatial domain of F by half the extent of the spatial domain of G, so that it the result well defined near the edges of the imager, and then cropping back to the domain of F after the operation. In Matlab, this is implemented by `convm(F, G, 'same')`. The temporal and other parameter-based convolutions are similarly defined, but are not used in this thesis.

Let disk(r) be the function that returns a normalized, discrete 2D disk filter kernel with radius r pixels as a video with two parameters, x and y (that is, the *result* has two parameters). Radius

[1] Those familiar with functional programming languages may recognize this as a *curried* function.

r may be any real-number. Since the output is discrete, let the disk be defined as an image of an anti-aliased disk with unit integral. When $r < \frac{1}{2}$, the disk becomes a impulse δ that is one at $[0,0]$ and zero elsewhere. Note that convolution with an impulse is the identity and convolution with a disk is a uniform blur. In Matlab, this video is produced by `fspecial('disk', r) / (pi*r*r)`.

4.5 Conversion to Matrices

$\mathbb{I}$ is the identity matrix (the bold font is to avoid confusion with the italic I used for images). $\mathbb{I}_n$ is the $n \times n$ identity matrix. Recall that ${}^{r}\mathbb{I}_c$ is the generalized identity previously introduced by equation 3.3 for parallel projection.

Let $M_{i,j}$ be the element at row i and column j of matrix M (the subscript on $\mathbb{I}$ has a different meaning, but it is never necessary to index into the identity matrix). Note that matrices are indexed exclusively by subscripts to distinguish them from multi-parameter video functions.

As previously mentioned, it is frequently useful to operate on the values in a multi-parameter video using linear algebra. Although it is natural to define the coercion of a video to a matrix by simply letting $M_{i,j} = I[i,j]$, in practice that is not useful. Instead, videos are unraveled into long vectors.

A vector hat denotes a multi-parameter video unraveled into a column vector along its dimensions in order, such that

$$\vec{F}[x + W((y-1) + H(\lambda - 1))] = F[x,y,\lambda], \tag{4.4}$$

where F describes an image with $W \times H$ pixels and uses 1-based indexing. For example, say that F represents the image on a 2×2 imager with three color samples. Let its values, expressed as a

three-dimensional array with integer λ, be:

$$F[\lambda = 1] \quad : \quad \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$F[\lambda = 2] \quad : \quad \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

$$F[\lambda = 3] \quad : \quad \begin{bmatrix} 9 & 11 \\ 10 & 12 \end{bmatrix}$$

The unraveling, $\vec{F}$ is the vector $[12436879101112]$. In Matlab, the unraveling is accessed as `F(:)`.

Because an unraveled image is a matrix and not a function, it is indexed by subscripts. Linear algebra operations like matrix-vector multiplication, inverse, and transpose operate normally on these vectors.

It is often useful to create a diagonal matrix from an unraveled image or other vector. Let x be a vector of length n. Let $X = \text{diag}(x)$ be the diagonal matrix such that

$$X_{i,j} = \begin{cases} x_i, & i = j \\ 0, & i \neq j \end{cases} \tag{4.5}$$

With this notation in hand, many of the operations in subsequent chapters can be expressed succinctly and clearly.

In closing, a thought about generality. Multi-parameter video typically has evenly sampled values along each parameter, and all samples at all parameter-points are gathered. But one can easily imagine a much less structured version with uneven sampling along parameters so that the result is not a regular grid in sampling space. As a motivating example, consider the video captured by 1000 parents at a graduation ceremony, where all are far enough form the stage that their videos can be considered to have a common center of projection. The resulting collection of videos differ along many parameters: temporal offsets (because they did not synchronize their start times), color-response curves, focus, focal lengths, apertures, etc., but not in any regularly sampled way (Wang and Yang [90] collect precisely this kind of irregularly sampled data). Both collections should still be describable as a multi-parameter image. I don't investigate the irregularly sampling further in this thesis, but point to it as an interesting area of future work.

Chapter 5

Related Work

Computational videography is one of many names given to the field of producing new or better output video streams from sets of input streams using a computer. This chapter reviews models of the image formation process, systems for capturing image sets and the processes for calibrating them, and then some of the hard problems in the field. These problems are hard in an empirical and not provable sense–although efficient and robust solutions *may* exist, they have not yet been discovered in the half century over which these problems have been active research.

For each problem the goal or ideal, often unacheivable, result is discussed and then previous work is presented and evaluated. Novel contributions in this thesis over previous work are discussed in the chapters that concerned with those contributions.

5.1 Capture Systems

Generalizing and extending previous work with multiple views of a single scene is the motivation for the multi-parameter approach. Previous systems have relied on a variety of physical optics techniques for creating multiple views. The most popular of these are beam splitters, which include polka-dot mirrors and half-silvered mirrors; a pyramid of mirrors with tip on the optical axis; and arrays of side-by-side cameras. An array of microlenses or micro-mirrors with a single imager is an inexpensive way of implementing the camera array method. Table 5.1 summarizes a representative subset of the prior work that uses these methods.

Design	Example	Advantages	Drawbacks
Dense Array of side-by-side sensors	[38] [3] [91]	Easy to calibrate (planar scenes). Inexpensive to prototype. Well suited for parallel processing.	Parallax occlusions. Only suitable for planar scenes.
Per-Pixel Filters	[8] [61]	Easy to manufacture. Easy to calibrate. Automatically synchronized.	Cannot virtualize lens or aperture. Filter tiles require interpolation. Hard to virtualize imager depth.
Mirror Pyramid post-lens	[3]	Compact; length not proportional to splits.	Asymetric PSF; not suited to defocus experiments. Cannot virtualize lens or aperture. Distorted projection requires warping.
Pre-Lens Splitter Tree	[62]	Easy to calibrate Compact. High quantum efficiency.	Cannot virtualize lens or aperture.
Post-Lens Splitter Tree	[26]	Can virtualize all parameters. High quantum efficiency.	Large. Hard to calibrate.

Table 5.1: Optical mechanisms for creating multiple copies of a view.

5.1.1 Beam Splitters

Prisms and half-mirrors are popular beam splitting mechanisms. They can be constructed to split light into two or more paths, and the ratio of intensities directed to each path at each wavelength can be adjusted. The cheapest and most common element is a half-silvered mirror. A drawback of mirrors is that their orientation must be calibrated relative to the optical path. In contrast, sensors placed immediately against the sides of a splitting prism are automatically registered up to a 2D translation. In the case of 3-CCD cameras, a dichroic prism is often used to capture three copies of the image, each representing a different spectral band. Prisms have also been used for high dynamic range imaging [44].

In the SAMPL camera implementation from this thesis, the beam splitters are placed between the lens and the scene. This allows the use of separate objective lens parameters (e.g., aperture size) for each sensor. It also provides enough physical room in the splitting tree for us to introduce many splits. An alternative is to split the light in between the lens and the sensor [62]. That alternative shares a single lens over all sensors, which simplifies lens calibration and reduces lens cost while

making calibration and filter changes more difficult.

5.1.2 Pyramid Mirrors

Another interesting way to create multiple copies uses a pyramid mirror placed behind the lens [41, 3]. This is a nice solution because it creates a very compact optical path. However, it has some drawbacks. It requires a large aperture, which leads to a narrow depth of field and limits the situations to which it may be applied (in the limit, it is impossible to create a duplicated pinhole view using a pyramid mirror). It is also non-trivial to divide light intensity unevenly between the image copies, as might be desirable for HDR. Furthermore, the edges between the individual mirrors cause radiometric falloffs as discussed in [3]. Such fall-offs, even when calibrated for, translate to a measurable loss the effective dynamic range of each copy.

When a pyramid mirror is placed behind the lens, the point spread functions due to defocus wedge-shaped instead of disk-shaped because each sensor's effective aperture is a wedge. This makes it difficult to fuse or otherwise compare images in which some objects are out of focus or where the different image copies are captured with different focus depths. Objects outside the depth of field appear not only defocused but also shifted away from their true positions.

5.1.3 Arrays and Alternatives

For scenes that are at infinity, there is no parallax and the optical centers of the sensors need not be aligned so long as the optical axes are parallel. In this case, view dependent effects and occlusions are not a problem for the imager. In practice, the parallax error may be tolerable for scenes as near as 10 meters provided the depth range of the scene is not too large. In this case, a stereo or other dense array of side-by-side sensors, e.g., [38, 91], can be used to obtain multiple copies of the scene as if they were captured from the same viewpoint. Compared to a beam splitter system, the cameras in a dense array receive much more light. However, a beam splitter system can operate over a larger depth range and offers the possibility of sharing expensive optical components like filters over multiple sensors.

One could use a mosaic of sensors to sample multiple parameters in a single image. The classic

Bayer mosaic tiles single-pixel band-pass filters over a sensor, capturing three wavelengths with a single monochrome sensor. Recently, filter mosaics have also been proposed for capturing other imaging dimensions with high precision (see [63, 65]). The benefit of this approach is that it can be implemented compactly and requires no calibration (once manufactured), making it ideal for many applications. The drawback is that it trades spatial resolution for resolution along other imaging dimensions, which is not desirable for some applications. It is also difficult to experiment with aperture, spatial resolution, and timing effects in such a system, all of which are explored in this paper.

5.2 Matting

Matting or "pulling a matte" is a computer graphics technique used in the film industry for separating the pixels of a *foreground* object from the pixels of the *background* in an image. The foreground can then be *composited* over a novel background to simulate the appearance of the object in the novel scene. The image is typically a frame from a video sequence, where a time-coherent matte is desired for the entire sequence. The practice of matting and recompositing against a novel background is extremely common–it was first used commercially in the 1950's on the film *Ben Hur* (1959) and now almost every feature film uses digital matting and compositing techniques.

Matting is a valuable technology. Chuang argues persuasively [18] that visual effects are key to the financial success of a film, and points out that the best-selling films of all time would not have been possible without matting. In addition to being a core technology for the billions-of-dollars entertainment industry, matting has applications in medical imagery and remote sensing where it is necessary to separate data from multiple depth layers for analysis. Many computer vision algorithms begin by segmenting an image into discrete objects at a pixel level. Matting solves the same problem (albeit typically for only two objects) at a sub-pixel level. Another application for matting is thus extending vision algorithms that use segmentation to sub-pixel precision.

The ideal matting algorithm takes as input a color video stream I_{rgb} and outputs three streams: a color, foreground-only stream αF_{rgb} with premultiplied alpha; a color, background-only stream B_{rgb}; and a single-channel stream of the partial coverage α of each pixel by the foreground object.

The ideal algorithm places no constraints on the input video. This means that both foreground and background are *dynamic* (contain moving objects), they may overlap at locations where there is no texture and identical color, the background contains objects at multiple depths, the camera may move, and the illumination is natural (uncontrolled).

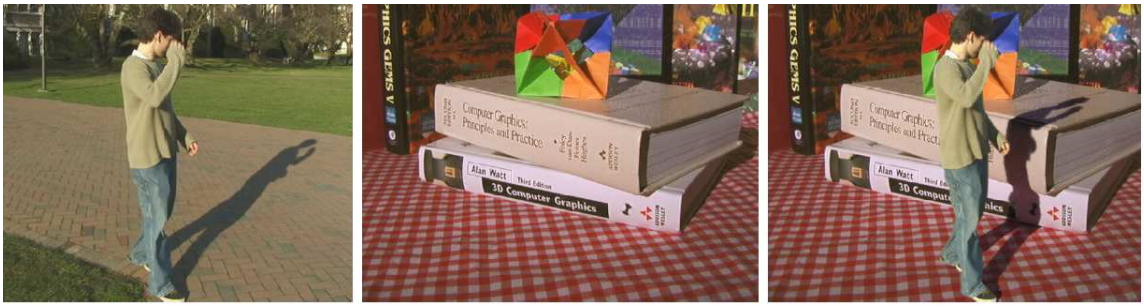
The ideal matting algorithm is unachievable in practice because too much scene information is lost when creating a discrete 2D video of a continuous 3D scene. This was proved formally by Smith and Blinn [82], and is readily apparent from consideration of the ambiguous cases. For example, where foreground and background are visually similar and have no texture, there is no distinction between them in a single frame. This ambiguity makes it impossible to discriminate the objects in all cases. Even in a scene with texture, this ambiguous case can occur where objects are under-exposed and appear black or are over-exposed and appear white.

A second reason the ideal matting algorithm is unachievable is that portions of the background may be entirely occluded by the foreground. There is no way to perfectly reconstruct these occluded areas and a real algorithm can only apply heuristics and interpolation to approximate them.

Even if the ideal matting algorithm were achievable, decomposition into foreground, background, and matte is insufficient for the ideal *compositing* result because of interreflection. The foreground and background objects affect each other's images because shadows and reflected light form a mutual environment. *Environment matting* and its extensions by Zongker et al. [96] and Chuang et al. [22] are methods for pulling mattes of transparent and reflective objects like glass against known, structured light backgrounds.

Shadows are another way that foreground and background interact. When reconstructing the background, it is often desirable to remove the shadow of the foreground object. Likewise, one might wish to film a foreground against one background, pull its matte, and then re-light the foreground with the environment of the new scene into which it will be composited. This has been explored extensively by Paul Debevec's group at ICT (see e.g., Debevec et. al [26]). One might also wish the foreground to cast shadows on the new scene, as explored by Petrović et al. [72], Finlayson et al. [34], Chuang et al. [21], and Cao and Shah [16]. Figure 5.1 shows a result from Chuang et al. [21] for a shadow matting problem with a static background, camera, and light source with a

dynamic foreground. The background is scanned with a shadow-casting stick to recover the effects of geometry on shadow displacement. The foreground and matte are extracted with Chuang et al.'s video matting [19] and then foreground and displaced shadow are composited into a novel scene.



(a) Source frame (b) Target frame (c) Composite

Figure 5.1: Shadow matting example from Chuang et al. [21].

This thesis limits the discussion of matting to the classic problem of separating foreground and background at sub-pixel precision, i.e., the 2D problem. It does not consider or further review the larger 3D scene reconstruction, relighting, and compositing problem.

Matting is a hard problem. This follows from the informal argument that one cannot achieve the ideal algorithm because there is insufficient information in the input. The formal reason that it is hard is that the problem is mathematically underconstrained, which is shown later in this chapter. The difficulty of the matting problem is also evident from the history of progress in solving it. Matting and compositing have been active areas of research since the 1950's, yet today there is no previous system that can autonomously pull a matte without a severely constrained scene. The apparent success that the film industry has enjoyed with matting has only been achieved by institutionalizing the constraints of the blue-screen matting algorithm. Sound stages are equipped with giant colored screens and the film crew actively works to avoid the many scenes for which that algorithm fails.

Matting and compositing problems are posed for scenes with well-defined, even if hard to discriminate, foreground and background objects. Assume these distinct objects of comparatively narrow depth extent, so that they can be well-approximated as planes parallel to one another and the image plane and and perpendicular to the optical axis. Let the image of the foreground plane against

a black background be αF and the image of the scene background without the foreground be B. Let α be an image where each pixel represents the partial coverage of that pixel by the foreground object– it is essentially an image of the foreground object painted white, evenly lit, and held against a black background. Note that the scale and resolution of the foreground and background images in world space differs. Due to perspective foreshortening, the pixels of αF and α correspond to a smaller world space area than those of B.

Porter and Duff's classic paper on compositing [73] formalized and standardized the notions of an alpha matte, pre-multiplied alpha, and the algebra of composition. They show that for a pinhole camera, the image of αF in front of ("over," in their notation) B is given by linear interpolation:

$$I_P = \alpha F + (1 - \alpha)B \tag{5.1}$$

Porter and Duff's definition of α and compositing rule holds only for sub-pixel coverage. That is, the foreground object is opaque and partially occludes each pixel due to motion blur (sub-exposure coverage), or because its silhouette lies within the pixel (traditional sub-pixel coverage). It can be extended to model defocus (sub-aperture coverage of the pencil of rays from the pixel) by blurring the images and changing the interpolation parameter at each pixel. Although α is frequently abused in real-time rendering as an approximation to colored transparency, i.e., filtering, that is a different phenomenon and is not accurately modeled by linear interpolation.

It is important to note that the variables in equation 5.1 have linear ($\gamma = 1.0$) radiance units (W/m^2). The pixel values stored in a video frame buffer and captured by many digital cameras are commonly brightness or voltage values, which can be approximated as non-linear ($\gamma \approx 1.2$) radiance values. I suspect that a common but unreported source of error in many matting results is a failure to convert to a linear space before matting. Doug Roble reports that this matches his experience at the Rhythm and Hues studio working with experimental matting techniques [78].

Matting is underconstrained as posed. Recall that the goal is to recover α and αF (and B, although the background is frequently discarded) given the composite I_P. Under the pinhole composition model in equation 5.1 there are seven unknowns ($F_r, F_g, F_b, \alpha, B_r, B_g, B_b$) and only three constraints (I_{Pr}, I_{Pg}, I_{Pb}), so there is not a unique input scene for each composite image. A matting algorithm must therefore constrain the problem.

Some matting approaches make matting tractable by introducing assumptions. Examples of useful assumptions are: the background is blue, the background is low-frequency, the edges of the foreground are within a narrow region, and the background is known. Other matting approaches make the problem tractable by gathering more data, for example, an infrared color channel or images of the foreground against two different backgrounds. The new defocus difference matting and defocus difference matting algorithms presented in this thesis use the data-gathering approach and operate on sets of images that differ by defocus. Another way to solve an underconstrained problem is to appeal to the user. *Interactive* or *user-assisted* matting solutions pair an algorithm with a painting tool. In such systems, a human being creates an approximate solution that is refined by the algorithm, and then corrects any artifacts in the final matte. Matting began with the rotoscope, which was an entirely user-driven solution, and evolved as researchers and engineers sought to reduce the user's burden.

5.2.1 The Rotoscope

The *rotoscope* is a physical device for projecting live action footage onto a hand-animator's canvas so that filmed objects can be traced and overdrawn, creating cartoons with realistic animation. The projector and camera for capturing the final animation share a single lens to ensure perfect alignment.

It was invented and first used by animator Max Fleischer in his *Experiment #1* film in 1915 and later patented in 1917 [35]. The same device was later used for manually tracing and hand painting individual binary alpha mattes for each frame (e.g., in *2001: A Space Odyssey*). Foreground and background can then be combined according to Porter and Duff's compositing rules using the hand-painted matte.

The obvious drawback to rotoscoping is that it requires significant time on the part of the animator. Working entirely in an artistic domain also limits the physical accuracy of the matte (it may be appropriate for visual effects, but not necessarily for medical and scientific applications) and it is impractical to manually pull mattes with fractional α values, such as for smoke or wisps of hair.

Manual rotoscoping is still in practice, although today it is performed digitally, with software

aiding the user by interpolating the matte between key frames. It is joined by digital blue-screen matting as the current best practices for matting in the film industry.

5.2.2 Self Matting

It is possible to directly photograph an object and its matte using two cameras, two sets of lights, and a background curtain. The first set of lights illuminates the object only. They are filtered so that their spectrum contains no energy at a specific wavelength, λ. The second set of lights uniformly illuminates the background only. They are filtered to contain only energy at λ. The cameras share an optical path using a beam-splitter and are co-registered but have different wavelength responses due to color filters. The first camera has a filter that absorbs λ and transmits all other visible wavelengths. Through this camera, the background appears black and the foreground is illuminated. The second, "matte" camera has the inverse filter. Through the matte camera, the foreground appears black and the background is uniformly bright. The matte is thus the inverse of the video stream captured by the matte camera. The color camera captures the pre-multiplied color directly because the background is entirely black in its view. This process is called *self-matting* because the scene produces its own matte and no further work is required to solve the matting problem. Smith and Blinn also call this a *multi-film* approach [82].

The first self-matting system was by Petros Vlahos. His system [87] lit the scene with a sodium lamp that naturally produces illumination at a single wavelength within the visible spectrum. One drawback to the sodium lamp approach is that the color camera and color illumination must be filtered. Using matte illumination outside the visible range (i.e., infrared and ultraviolet) avoids this problem and has long been used (e.g., Vidor's 1960 IR system [86]) to simplify the filtering process. Self-matting techniques are still active in film production and research–see Fielding's text [33] for a description of those used in cinematography.

A recent research system by Debevec et al. [26] performs self-matting and compositing. Their system uses visible and infrared cameras connected by a beam splitter to pull high-quality mattes in a controlled studio environment. The background is a special cloth material chosen which appears black under visible wavelengths and is highly reflective ("white") under infrared light for an infrared

camera. Flooding this background with infrared light therefore creates the image of a reverse matte in the infrared camera, where background is white and any foreground object before it appears black. The foreground is lit by a dome of LEDs, which are controlled by a computer to match the lighting environment of the scene into which the foreground will later be composited. Because color LEDs have little emission in the infrared portion of the spectrum, the lighting on the subject does not corrupt the matte.

The obvious drawback to self-matting processes is that the lighting and background must be carefully controlled. Thus it is inappropriate for shooting outdoors (natural lighting), on location, or in scenes with large backgrounds, and for consumer-level film production.

The sodium lamp technique was used on *Mary Poppins* (1964), but appears in few other major films [78]. Infrared and ultraviolet mattes are used occasionally for objects where blue-screen matting fails; e.g., multi-colored or reflective surfaces. In general, however, filmmakers are averse to techniques that divide light with beam splitters (a drawback shared with my system) and find the process of using specific filters and backgrounds with specific reflectivities cumbersome. Infrared shares these problems and is also believed to be blurry and noisy compared to higher-frequency visible light [78].

5.2.3 Blue-Screen Matting

After his sodium lamp work, Vlahos invented the now-famous technique that won him his second technical Oscar award: *blue-screen matting*. In a blue-screen system, a single camera films a foreground object that contains no blue against a uniformly lit blue screen background. Because B is known, occupies a single color channel, and has little overlap with the foreground in color space, the matting problem is sufficiently constrained to solve for plausible values of α and αF from a single image.

Although it is not explicitly mentioned in the literature, there is some defocus error in blue-screen matting. Blue-screen matting creates “pinhole mattes;” that is, the output is intended for composition by Porter and Duff’s pinhole “over” operator. However the matte is pulled using input from lens cameras. Although the resulting composite image is not physically plausible (the α

channel and defocus will be slightly incorrect), the mismatch is rarely visually apparent. The observation that defocus can be approximated as pinhole linear compositing [5] underlies many other vision systems, including the new defocus matting algorithms discussed in this thesis.

When Vlahos' patents began to expire, Smith and Blinn introduced [82] blue-screen matting to the research community and provided a detailed analysis of Vlahos' methods. Vlahos' original blue screen method is described in a 1971 patent [88]. It computes the alpha and premultipled foreground as:

$$\alpha = \max(0, \min(1, 1 - k_1 (I_b - k_2 I_g))) \tag{5.2}$$

$$\alpha F_{r,g} = \alpha I_{r,g} \tag{5.3}$$

$$\alpha F_b = \min(\alpha I_b, k_2 I_g) \tag{5.4}$$

where k_i are scene-specific constants set by a user. Increasing k_2 improves selectivity at the expense of blue-spill; it is typically on the range $0.5 < k_2 < 1.5$ [89]. The "no blue" requirement for the foreground object is more formally given as color channels related by $F_b \leq k_2 F_g$. The user interactively adjusts the constants based on the scene until the matte is visually attractive.

For such a simple system and algorithm, the blue screen process has proven remarkably easy for users to tune, is robust and temporally coherent, and gives good fractional alpha values at the borders of the foreground. The color restrictions intentionally avoid shades of gray and human flesh tones, which are the most likely colors for the foreground object in film production.

In subsequent patents [89, 23], Vlahos and his colleagues at Ultimatte refined the formula. These refinements targeted a narrower area of the color space and addressed the major sources of artifacts: *blue spill* (reflected blue light on the foreground object and around the edges), *backing shadows* (foreground shadows on the background that should be considered part of the foreground object), and *backing impurities* (arising from an imperfectly blue background screen).

The blue screen technique has since been applied with other uniform background colors including orange, black, and green. Green is the most popular for digital video (although it is common to refer to all colored-screens as "blue screens" in deference to Vlahos' original method). This is primarily because the Bayer pattern of a digital camera has double resolution and sensitivity in the

green channel [8]. It is also fortunate for daylight and other full-spectrum illumination, which tends to have more energy in the center of the spectrum where the green wavelengths lie.

In practice, blue-screen matting requires more than a uniform-color screen–the screen must be carefully lit to avoid shadows, maintain uniform intensity, and avoid reflecting blue light on the subject. Of course the actor, lighting, and costume are restricted to avoid the uniform color of the background. This means no blue eyes or blue jeans!

Blue-screen matting is so popular that artifacts due to violating these restrictions can occasionally be observed during a television news cast. A weatherman commonly stands in front of a blue background and the weather map is digitally composited in behind him. When the weatherman wears a blue or green tie, the tie may disappear in the final composite and appear as a hole in his chest. If the weatherman touches the screen and is poorly lit his shadow will be a darker shade of blue than the rest of the screen and be non-zero in the matte. Instead of a shadow cast on the composited map, the shadow then appears as a blue shape superimposed on the weather map.

Because colored screens are the best-of-breed practical solution for film production, the industry has adapted its techniques and training to accommodate them. Sound stages are well equipped with pull-down screens. Lighting, costume, and camera operator professionals are well-educated in their use to avoid artifacts [78]. Thus the weatherman's occasional artifacts are almost never observed in feature films where there are many takes and sufficient editing time to remove footage with artifacts. In 1995, it was estimated that 70% of a ChromaKey [58] operator's time is spent retouching mattes to correct artifacts [60].

5.2.4 Global Color Models

Smith and Blinn [82] note that blue screen matting algorithms create a ternary partition in the three dimensional space of all RGB colors. The three RGB subspaces produced by this partitioning are the background subspace where $\alpha = 0$, the foreground subspace where $\alpha = 1$, and the fractional subspace that lies between them. Within this fractional subspace, the value of α is linearly interpolated between the foreground and background subspace. Such a partition can also be viewed as creating a mapping from (R,G,B) colors in the input composite directly to α values in the matte.

Smith and Blinn call the RGB partitions *separating surfaces*. For blue-screen matting, the separating surfaces are polyhedra close to the $b = 1.0$ plane. Because the separating surface is based on a global color assumption (e.g., the background is blue everywhere), blue-screen matting is considered a *global color model*. Several other global color models exist.

The simplest global color model is the threshold or *luminance matte*. It assumes a bright foreground and dark (black) background. It then chooses the matte and pre-multiplied foreground as

$$\alpha = k_r I_r + k_g I_g + k_b I_b > \theta \tag{5.5}$$

$$\alpha F = \alpha I \tag{5.6}$$

for a scene-specific intensity threshold $0 < \theta \approx 0.25 < 1$ and color space constants $k_{rgb} \approx (0.3, 0.6, 0.1)$. In the case of luminance matting the separating surface is simply a plane through RGB space with normal k_{rgb} and offset θ.

Luminance matting can also be viewed as *black*-screen matting, since it assumes $B \approx 0$. Luminance matting is rarely used in film production today as a matting algorithm; however, it is a common trick exploited by artists in paint programs like Photoshop to quickly strip a background.

Several global color model matting algorithms have been developed as derivatives of the original blue-screen matting, such as ChromaKey, the Ultimatte, and Priamatte. These are all variations on the colored-screen idea. The advantages they offer over Vlahos' original colored screen arise from better discrimination through a more detailed separating surface. For example, Mishima's *ChromaKey* [58, 59] begins with the two blue-screen polyhedra in color space centered around the mean of the (user-specified) background color distribution. The user interactively moves the vertices to grow the foreground partition and shrink the background partition as much as possible. If the background actually is a uniformly lit colored screen, the background partition will shrink to a point and the foreground partition will grow to the largest possible polyhedron that does not enclose any observed foreground color. However, if the background has a wider distribution, its partition will have non-zero area. As with other global color models, α is defined as the normalized distance of a given input composite image pixel between the foreground and background partitions in color space.

The lighting and scene restrictions of global color models are unacceptable for on-location filming and the potential artifacts make it undesirable for professional live television applications and for use by amateurs on home movies. To approach the ideal matting algorithm, more sophisticated methods have been developed. These address the *natural image matting* problem, where the background and lighting are uncontrolled. Specifically, these new methods are appropriate for scenes where the background color distribution is heterogeneous and is therefore not well-approximated by a global color model.

5.2.5 Local Color Models

A global model of the separating surface means that the separating surface is invariant to pixel position. This naturally restricts global models to operating on scenes where the global color distributions of foreground and background are distinct. In images of natural scenes, the *global* color distributions of foreground and background commonly overlap. The distinction between objects is a local one. For these scenes, foreground and background distributions are distinct only within small neighborhoods. To address the natural image matting problem it is necessary to allow the surface to vary between pixels as the background color changes, forming a *local color model*.

Difference matting (also known as *background subtraction* and *video replacement*) uses a straightforward local color model. Assume that the background is static throughout a sequence, i.e., it does not move, the lighting does not change as the foreground moves, and that the camera is static. Photographing the background before the foreground object is present in the scene directly produces the background image B. This is a perfect local color model for the background; at each pixel, the precise background color is known. From this color model, a simple matte can then be pulled by the observation that α is small where $I \approx B$ and large elsewhere. Formally,

$$\alpha = \min(1, \max(0, k \sum_{rgb} |I - B|)), \tag{5.7}$$

where scene-dependent constant $k \approx 20$ is chosen based on the actual difference between foreground and background color distributions.

Qian and Sezan's method [76] extends the basic difference matting approach with a more sophisticated probabalistic model on normalized color data. The results of this model are rounded

to a binary $\alpha = 0,1$ matte. To fill holes and produce fractional alpha values, they introduce an anisotropic diffusion model at the border between foreground and background.

Difference matting assumes a known, constant background. This limits its application. For scenes with dynamic backgrounds the assumption fails completely and a method for forming a local color model without such perfect information is required.

5.2.6 User-Assisted Local Color Models

For other methods that use local color models more sophisticated than difference matting, the background is unknown (not blue, not static) and the camera is not static. Without those constraints, the matting problem is again underconstrained and the major challenge is forming a good local color model using only the input composites. Matting techniques that use local color models must therefore introduce additional knowledge or assumptions about the background to give a unique solution. Because human operators are very successful at solving the matting problem (e.g., with a rotoscope), a good source of this additional information is a user. A successful strategy for matting is to have the user paint a coarse matte for an image and then refine that matte using an algorithm. User assistance casts matting as a machine learning problem. In this context the coarse matte is the training set and the separating surface is a learned model of local background color statistic. Because of the need for user input, the learning approaches are considered *tools* instead of closed systems or algorithms [80].

Tools are often employed within an iterative workflow. The user steers the algorithm by repeatedly modifying the training set until the matting system learns the correct model. It is common to provide the user with a set of tools for painting corrects into the final matte where the algorithm still produces artifacts, as it is often easier to fix a few small errors manually than reverse engineer the ideal training set.

The hand-painted coarse matte is called a *trimap* segmentation because it partitions the the image into three mutually-exclusive and collectively exhaustive spatial regions. The three regions are named *known background*, where the user asserts that $\alpha = 0$; *known foreground*, where the user asserts that $\alpha = 1$; and the *unknown region* in which the alpha values are unknown and likely

fractional. These regions are frequently represented as sets of (x,y) locations written Ω_B, Ω_F, and Ω and manipulated with set operators. The vertical bar operator, $e|S$, binds loosely and used to express a relation that holds for expression e on set S. It is a compact notation for $e|_{xy \in S}$. The horizontal bar is set inverse, so that $\bar{\Omega}$ and Ω are mutually exclusive and collectively exhaustive.

Note well that the foreground, background, and unknown *regions* of a trimap are a partitioning of XY positions on the input composite. They are a completely distinct concept from the similarly named foreground, background, and fractional *RGB subspaces* created by separating surfaces.

Let T be the scalar trimap image that is painted by the user with three values such that

$$T = \begin{cases} 0 & | \quad \Omega_B \\ \frac{1}{2} & | \quad \Omega \\ 1 & | \quad \Omega_F \end{cases} \tag{5.8}$$

Boundaries are also written in set notation, using the tightly-binding partial operator, ∂. Let $\partial\Omega_F$ be the boundary between Ω_F and Ω, $\partial\Omega_B$ be the boundary between Ω_B and Ω, and the entire boundary of Ω be

$$\partial\Omega = \partial\Omega_B \cup \partial\Omega_F \tag{5.9}$$

Assume that Ω_F and Ω_B never meet directly. Figure 5.2 shows a trimap painted in Photoshop for an image of an actor's head and shoulders with the regions and boundaries labeled. The black region is the known background and the white region is the known foreground. The gray region between is the unknown. Incorrectly classified pixels in the trimap cause incorrect training, but it is desirable for the user to be able to paint the trimap quickly, without tracing around every feature of the silhouette. It is therefore common practice to paint a very wide unknown region as shown, even though the actual fractional alpha values may lie within a narrow strip. In the figure, the red lines indicate the boundaries between regions.

The current best model for learning a separating surface is Chuang's Bayesian matting [20, 19]. His significant matting thesis [18] compares the four major learning models in detail. Figure 5.3, adapted from that thesis, summarizes the main differences between the Ultimatte corporation's Knockout algorithm [10, 11] by Berman et al. (packaged commercially as Corel Knockout), Ruzon

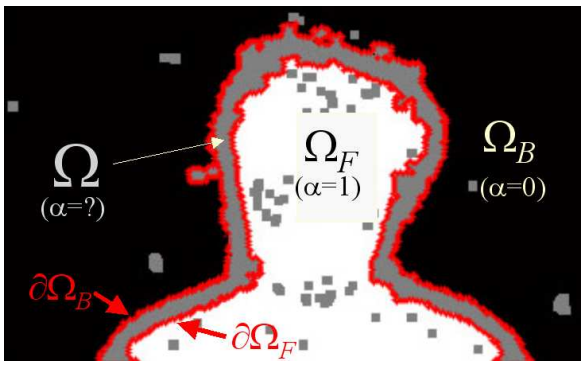


Figure 5.2: Sample trimap and notation.

and Tomasi's algorithm [80], Hillman's algorithm [43], and Bayesian matting. Misihima's ChromaKey [59] blue-screen matting algorithm is shown at the far left for comparison. The top row of subfigures depicts a small area of trimap in XY image space. The gray band through the center of each is Ω. Ω_B is to the right in black and Ω_F is to the left in white. The bottom row of subfigures depicts the separating surfaces learned by each model. The surfaces are shown in a 2D slice of RGB color space and are intended to be abstract rather than concrete examples (the images from which the surfaces have been learned are not shown in the figure).

Knockout computes the expected foreground color in a neighborhood around an unknown pixel as the weighted average of pixels on $\partial\Omega_F$. The nearest known foreground pixel receives the highest weight and weights fall off linearly with distance to a minimum of zero at twice the distance of the nearest foreground pixel. The expected background color is computed in a similar manner using $\partial\Omega_B$. In color space (Figure 5.3g), the local separating surfaces are through these expected points and perpendicular to the line between them. Variations on this basic approach, including more sophisticated projections in color space, are described in the patents [10, 11]. Knockout is packaged as a plug-in tool for Adobe Photoshop and is traditionally applied to still images only. The other methods are used with video, which is a harder problem because the mattes for successive frames must be temporally coherent. Knockout was the first of the learning models (although its learning method is a simple weighting). Compared to the more aggressive methods that followed, the contributions of Knockout are in being both the first and most practical learning method–the others have yet to see such widespread commercial usage.

Ruzon and Tomasi use a more probabalistic model than the explicit surfaces of Knockout. They were the first to use a sophisticated learned model. For each sub-region of Ω, they counstruct a bounding box that penetrates a small distance into Ω_F and Ω_B. The foreground and background colors in the box are treated as samples from unknown distributions. They fit an axis-aligned 3D (RGB) Gaussian mixture model to both the foreground and background distributions. Each of the foreground Gaussians is paired with a background Gaussian to form a probabalistic separating surface. Figure 5.3h shows one such pair. Heuristics based on coherence and angle are used for both the pairing and the color-space clustering into a small number of Gaussians. A third Gaussian mixture model is fitted to the unknown pixels; the α value is the one that yields a distribution with the maximum likelihood.

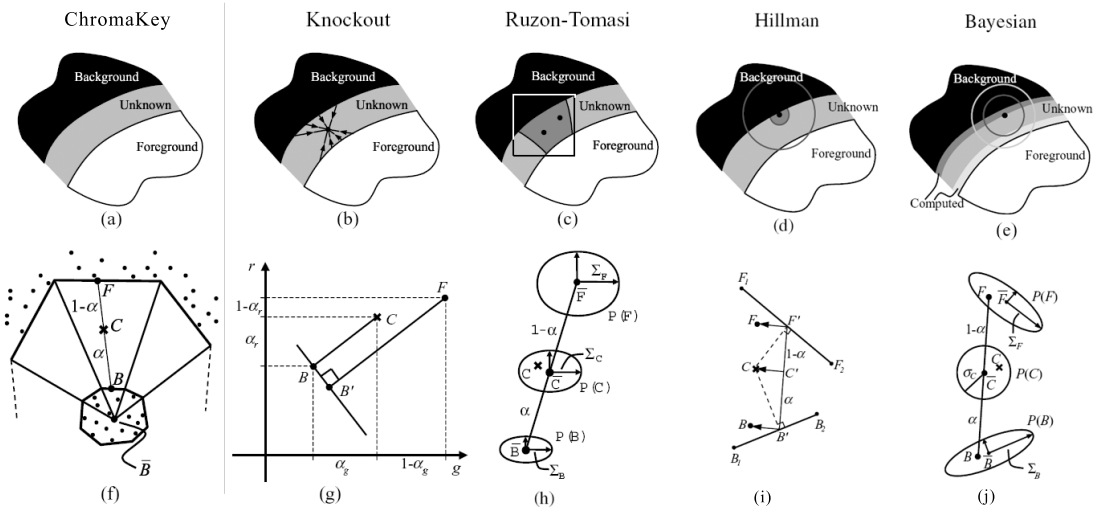


Figure 5.3: Learned separating surfaces from user-assisted matting algorithms.

Hillman observed that the Gaussians in Ruzon and Tomasi's method are typically long and skinny and simplified their model to oriented line segments fitted by principle component analysis (PCA). Allowing orientation better captures transitions between colors and darkening of colors due to lighting falloff, which is along the diagonals of RGB space. He also introduced the notion of pixel marching, where the unknown values immediately adjacent to $\partial\Omega_F$ and $\partial\Omega_F$ are first recovered, and the model is then re-evaluated to include that new data. Instead of a bounding box, the bounding region is a circle, which helps eliminate directional dependence in XY space. There is still some

order dependence because Hillman chose to march along rasters instead of strictly perpendicular to *∂Omega*. Foreground, background, and α are recovered in a similar manner to Ruzon and Tomasi.

Bayesian matting extends Hillman's ideas–pixel marching and oriented distributions–with a Bayesian learning model. A Gaussian model is used, as in Ruzon and Tomasi, but the Gaussians are oriented and learned by Bayes' rule. A simple sum-squares metric of the difference over clustered points in color space from their average gives the mean and covariance of the Gaussians for foreground and background distributions. The distributions are paired off. A point in Ω is tested against all pairs of Gaussians to recover α and the one with the highest maximum liklihood is chosen. A true mixture model would fit the point to all pairs and combine their results, but the simple exclusion test was found to be sufficient and is mathematically simpler. Like Hillman, Bayesian matting solves for a single pixel and then refines the color space model using that new value. Unlike Hillman, Chuang chose to march inwards from $\partial\Omega$ producing an orientation independent solution.

In order to produce video mattes, Chuang et. al [19] have extended Bayesian matting with a method for interpolating trimaps between keyframes approximately nine frames apart. This reduces the amount of user interaction required, however the user must still view the final sequence and introduce new keyframes to correct artifacts. Chuang has also used Bayesian matting to build algorithms that address other interesting matting problems including shadow matting [21] and more effective blue-screen matting [18].

It is hard to compare the learning based approaches in a fair manner because they are highly dependent on user input. If the user chooses a good trimap, all of these methods produce excellent results. The common mode of operation is for the user to refine the trimap until a satisfactory result is achieved. This is a strength of the learning approach, since the control is intuitive and visual.

5.2.7 User-Assisted Gradient Methods

A separating color model is not the only way to solve the matting problem. An alternative family of matting algorithms based on the partial derivatives of the composition equation has proven very successful. Like the learning methods, these gradient methods also employ user input in the form of trimaps to reduce the scope of the problem.

Mitsunaga et al.'s Autokey system [60] pulls a matte from an image and a trimap by making assumptions about the gradient and frequency content. Autokey assumes that the scene is low frequency. Specifically, it assumes that both F and B contain no high frequencies near a curve C. Curve C runs through Ω such that $\alpha = \frac{1}{2} \mid C$, making C mostly parallel to $\partial\Omega$. Curve C is oriented so that Ω_B is on the left and Ω_F is on the right as the curve is traversed (i.e., the background is outside a clockwise-winding). In Mitsunaga et al.'s original implementation, the user paints the C path directly and the system infers the foreground, background, and unknown regions, however it is easier to describe the system using the explicit trimap regions.

By AutoKey's low-frequency scene assumption, high frequencies in I within unknown region Ω can be attributed only to changes in α. These frequency constraints can be expressed as constraints on the gradients of the images:

$$\begin{aligned} \nabla F &\approx 0 \mid \Omega \\ \nabla B &\approx 0 \mid \Omega. \end{aligned} \tag{5.10}$$

The gradient of Porter and Duff's over composition is given by the chain rule:

$$I = \alpha F + (1-\alpha)B \tag{5.11}$$

$$I = \alpha(F-B)+B \tag{5.12}$$

$$\nabla I = \nabla B + (\nabla\alpha)(F-B) + \alpha(\nabla F - \nabla B) \tag{5.13}$$

Substituting the constraints from Mitsununga et al.'s low-frequency assumption equation 5.10 into the gradient of the composition equation, the expression for the composite image gradient simplifies to:

$$\nabla I \approx (F-B)\nabla\alpha \mid \Omega, \tag{5.14}$$

which gives a partial differential equation for the α gradient,

$$\nabla\alpha \approx \frac{1}{F-B}\nabla I \mid \Omega. \tag{5.15}$$

To simplify this to a scalar equation, Mitsunaga et al. reduce color images to grayscale. Any linear combination of color channels is sufficient, however, by projecting the difference $F-B$ onto

a color axis approximately perpendicular to the average difference over the whole image, AutoKey increases the signal-to-noise ratio. AutoKey further increases selectivity by computing weighing the gradient computed from this grayscale image. The weight factor is the dot product of the actual and expected α gradients, where the expected gradient is perpendicular to path P. This weighting effectively diminishes the gradient magnitude in areas that have small hue variation and are therefore likely entirely from the foreground or background. Let G be the single-channel projected, weighted difference image of F and B.

Given a path $P : p(t) = (x,y)$, α is related to its gradient $\nabla\alpha$ by:

$$\alpha(p(t)) = \alpha_0 + \int_P \nabla\alpha(x,y) \cdot dp(t). \tag{5.16}$$

Substituting the α gradient approximation from equation 5.15 and the enhanced image G for $F - B$ gives the following solution to the partial differential equation for α:

$$\alpha(p(t)) = \alpha_0 + \frac{1}{G}\int_P \nabla I(x,y) \cdot dp(t), \tag{5.17}$$

where α_0 is determined by the boundary constraints,

$$\alpha_0 = \begin{cases} 0 & | \quad \partial\Omega_B \\ 1 & | \quad \partial\Omega_F \end{cases} \tag{5.18}$$

Figure 5.4 shows mattes pulled for four frames by AutoKey (figure from [60], at approximately the same resolution as the original paper). The input composites, I, are on the top row. They show consecutive frames of a player in a sporting event. The player is viewed from behind with his yellow pants prominent in the frame. Result mattes, α, are on the bottom row of the figure. The red curves on the input images are piecewise cubic curves comprising C. The leftmost curve is drawn by the user, who has chosen to pull a matte where the player's pants are the foreground and the stadium and player's upper body are the "background." The curves in subsequent frames are propagated forward by the system through an assumption of temporal coherence. They are adjusted by the user before the final matte is pulled.

The AutoKey results are good for the sequence in figure 5.4. The mattes are clean, with sharp edges, and appear time coherent. As has been the case throughout the history of matting, the motivation for further research was not that previous results were poor, but that previous results were

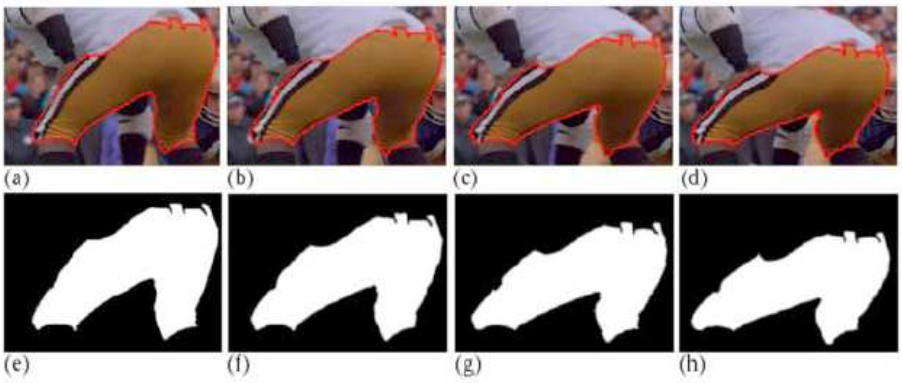


Figure 5.4: AutoKey matting result from Mitsunaga et al. [60].

excellent and it was desirable to achieve similar results for a larger class of scenes and with less user input. In the case of AutoKey, the next more general step was to obtain good results for scenes with fine features like hair and large areas of fractional α values. The Poisson Matting method [84] builds on AutoKey by first extending the gradient math and then adding trimaps to help solve it (although I have described it using trimaps, the original AutoKey implementation used directed parametric curves near $\alpha = 0.5$ as user input).

The low-frequency scene approximation (equation 5.15) can be expressed as a Laplace problem of the form

$$\alpha = \underset{\alpha}{\operatorname{argmin}} \int_{\Omega} ||\nabla\alpha - \nabla I||^2 dA \tag{5.19}$$

with Dirichlet boundary condition

$$\alpha = \alpha_0 \mid \partial\Omega. \tag{5.20}$$

where α_0 is given by equation 5.18 as before.

The advantage to casting the α equation as a Laplace problem is that this is a well-known partial differential equation form. The solution to the Laplace problem is given by Poisson's equation,

$$\Delta\alpha = \operatorname{div}(\nabla I). \tag{5.21}$$

Many Poisson solvers exist. For a small Ω, α may be obtained by directly solving a linear system. For ill-conditioned systems or large Ω, the Gauss-Seidel solver is frequently applied.

This Laplace/Poisson formulation is popular in computer graphics. It has been used by Fattal et al. [30] to create high dynamic range imagery, by Perez et al. [71] to seamlessly insert patches into images, and by Raskar et al. [77] to fuse day and night images of the same scene. As noted by Perez et al., there are many other seamless editing techniques (e.g., [15, 28, 51]) that are mathematically equivalent to solving the Laplace problem.

Sun et al.'s Global Poisson Matting [84] applies the Poisson formulation to matting. They further provide a suite of local tools to manually clean up the resulting matte, which are not discussed here. Poisson matting solves the Poisson problem for α on Ω, where the boundary conditions are given by the trimap, and the gradient is given by equation 5.13. That is, high frequencies in α come from the pinhole image and low-frequencies from the user-drawn trimap.

Standard practice for solving the Poisson equation is as follows. Laplace's equation constrains real-valued function α over region Ω to have boundary values given by L and gradient given by ∇I:

$$\nabla\alpha = \nabla I \,\big|\, \Omega \tag{5.22}$$

$$\alpha = \alpha_0 \,\big|\, \partial\Omega \tag{5.23}$$

That is, the low frequencies should come from L and the high frequencies from I. One cannot usually satisfy both constraints simultaneously, so this becomes a minimization problem. Let $E(\alpha)$ be the real-valued function measuring the squared error in the gradient of α compared to the desired gradient,

$$E(\alpha) = \int_\Omega ||\nabla\alpha - \nabla I||^2 dA \tag{5.24}$$

One can now solve the minimization problem on $E(\alpha)$:

$$\alpha = \underset{x}{\operatorname{argmin}} E(x) \,\big|\, \Omega \tag{5.25}$$

$$\alpha = \alpha_0 \,\big|\, \partial\Omega \tag{5.26}$$

Suppose there is a function h that is zero on the boundary of Ω and nonzero elsewere. Let $\alpha^\star$ be the unknown α that minimizes E and let $g(e) = E(\alpha^\star + eh)$, the error of perturbing the minimizing

solution by scalar e times function h. Since $\alpha*$ minimizes E, clearly for arbitrary h, $e=0$ minimizes g, so

$$\frac{\partial g(e)}{\partial e}\Big|_{e=0} = 0 \tag{5.27}$$

The partial derivative with respect to e is given by:

$$\begin{aligned}
\frac{\partial g}{\partial e} &= \frac{\partial}{\partial e}\int_\Omega ||\nabla(\alpha^\star + eh) - \nabla H||^2 dA && (5.28)\\
&= \int_\Omega 2(\nabla(\alpha^\star + eh) - \nabla I)\cdot\frac{\partial}{\partial e}(\nabla\alpha^\star + e\nabla h)dA && (5.29)\\
&= 2\int_\Omega \nabla(\alpha^\star + eh - I)\cdot(\nabla h)dA && (5.30)
\end{aligned}$$

Substituting into equation 5.27, at $e=0$:

$$\int_\Omega h\nabla(\alpha^\star - I)\cdot(\nabla h)dA = 0 \tag{5.31}$$

Integration by parts gives:

$$\int_{\partial\Omega} h\nabla(\alpha^\star - I) - \int_\Omega \nabla\nabla(\alpha^\star - I)\cdot h\, dA \tag{5.32}$$

The left-most term is zero because we assumed $h = 0|\partial\Omega$, so

$$\int_\Omega \nabla^2(\alpha^\star - I)\cdot h\, dA = 0 \forall h \tag{5.33}$$

This is true if and only if:

$$\nabla^2\alpha^\star = \nabla^2 I, \tag{5.34}$$

which is Poisson's constraint, frequently written as

$$\Delta\alpha^\star = \mathrm{div}(v) \tag{5.35}$$

where $\mathrm{div} = \nabla\cdot = \frac{\partial}{\partial x} + \frac{\partial}{\partial y}$, $\Delta = \nabla^2$, and $v = \nabla I$ is the *guidance vector field* (denoting that only the gradient of I is important).

With the Poisson and boundary constraints, solve a linear system for α that minimizes the sum-squared error. Assume discrete I, α_0, and α unwrapped into vectors of length n. Let m be the mask

vector for $\bar{\Omega}$ such that $m = 0 \mid \Omega$ and $m = 1 \mid \bar{\Omega}$. Let $M = \text{diag}(\vec{m})$ be the diagonal matrix formed from the unraveled mask. Using the mask, the constraints can be written with matrix products instead of set notation as

$$\begin{aligned} M\vec{\alpha} &= M\vec{\alpha}_0 \quad \text{and} \\ (\mathbb{I} - M)\nabla^2\vec{\alpha} &= (\mathbb{I} - M)\nabla^2\vec{I}. \end{aligned} \tag{5.36}$$

Let Λ be the matrix that computes the Laplace of a matrix unwrapped into a vector. Both M and Λ are size $n \times n$, but are sparse with only O(n) elements each. The solution to α is now the solution to the sparse $2n \times n$ system:

$$\begin{bmatrix} M\vec{\alpha} \\ (\mathbb{I} - M)\Lambda\vec{\alpha} \end{bmatrix} = \begin{bmatrix} M\vec{\alpha}_0 \\ (\mathbb{I} - M)\Lambda\vec{I} \end{bmatrix} \tag{5.37}$$

$$\begin{bmatrix} M \\ (\mathbb{I} - M)\Lambda \end{bmatrix} \begin{bmatrix} \vec{\alpha} \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} M \\ (\mathbb{I} - M)\Lambda \end{bmatrix} \begin{bmatrix} \vec{\alpha}_0 \\ \vec{I} \end{bmatrix} \tag{5.38}$$

Note that the constraints in equation 5.36 apply to different elements of the vectors since M and $\mathbb{I} - M$ are disjoint. This allows us to simplify the system to an $n \times n$ problem on sparse matrices:

$$(M\Lambda + \mathbb{I} - M)\vec{\alpha} = M\Lambda\vec{I} + (\mathbb{I} - M)\vec{\alpha}_0 \tag{5.39}$$

$$\vec{\alpha} = (M\Lambda + \mathbb{I} - M)^{\dagger}(M\Lambda\vec{I} + (\mathbb{I} - M)\vec{\alpha}_0). \tag{5.40}$$

For small Ω (2000 unknown pixels) and images (256×256), a modern CPU can solve this system in about 20 seconds. Bolz [14] reports great success at solving similar sparse problems even faster using graphics hardware. For large Ω the specialized Gauss-Seidel solver has been reported to give good results for this problem [84, 71].

5.2.8 Defocus

Thus far I have described work in the computer graphics literature that directly targets the matting problem. The general problems of scene reconstruction and object recognition are the core of computer vision. The new matting method presented in chapter 8 combines ideas from graphics like the

trimap and the α, F, B scene model, with ideas from vision like defocus and regularization. This subsection reviews the vision work that underlies the matting in this thesis.

A simple characterization of a defocussed object is that it is blurry. Chapter 3 showed that this is because the lens camera image of a defocussed plane of constant depth is the convolution of an image of the aperture with the pinhole image of the plane.

Both Asada et al. [5] and Bhasin and Chaudhuri [12] observe that the image of a defocussed plane in front of a focussed plane is similar but not identical to the image produced by blurring the near plane and then compositing. They examine the discrepancies between ideal optics, various Fourier approximations, and real images. Asada et al. propose an approximation called the reversed projection blurring for modeling defocus near object edges. From it, Asada et al. developed a method [6] for reconstructing object depth at sharp edges by comparing images with different planes of focus. Figure 5.5 shows the main result from that technique. The five images on the left are from a static scene captured with varying focus parameters. The scene contains two toy dinosaurs about 1m apart on a desk. On the right is an image in which depth discontinuities have been marked as black. Depths are computed at each of the black pixels. The results are very precise for the table edge, where the depth slope is nearly perfectly linear and the standard deviation is 3.5 mm from a fitted line. Asada et al. contrast this with previous depth from defocus approaches on a single image (e.g., [46])which tended to produce less stable output.

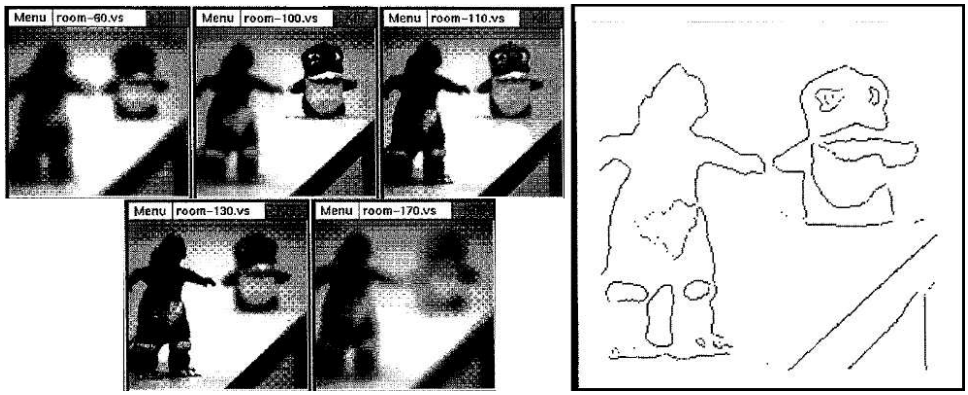


Figure 5.5: Depth edges from defocus by Asada et al. [5].

In general, the problem of depth-from-defocus is well studied [92, 64, 39, 17, 81, 62, 69, 29, 24].

The best video results are by Nayar et. al [62], who report better than 99% accuracy for 30 fps 512×480 video of scenes containing small polyhedral objects using active illumination.

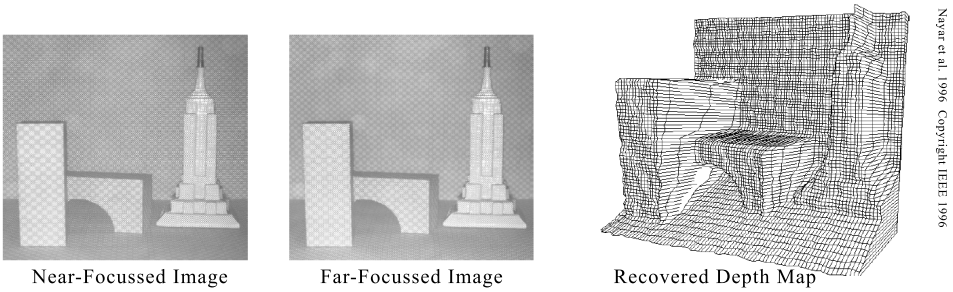


Figure 5.6: Depth from defocus video with structured active illumination by Nayar et al. [62].

Favaro and Soatto [31] created a depth-from-defocus scene reconstruction method that models a scene as a set of surfaces at different depths. Using a still camera, they capture a series of images focussed progressively deeper into the scene. Given that all images must come from the same underlying scene, they use a non-linear optimizer to search for a set of surfaces and depths that plausibly model the scene. To constrain the problem they assume that the background object is either wholly visible or wholly occluded at each pixel and that there is strong depth coherence. This is equivalent to finding a level set for the matte in the matting problem and producing a binary α result. In addition, they introduce regularization terms on the color channels that helps the optimizer converge to a plausible solution. Figure 5.7 shows a typical input scene and the capture system. The scene consists of a highly textured background containing the acronym “IEEE” and a highly textured foreground.

Figure 5.8 shows the reconstructed depth and color information for the scene. Although the scene is assumed to comprise a set of objects at different depths, the depth within an object may vary. As with other depth-from-defocus methods, the scene chosen is small in order to obtain large PSFs and is highly textured to aid in detecting defocus.

Schechner et al. [81] were the first to use a depth-from-focus system to recover overlapping objects with fractional alpha. They drive a motorized CCD axially behind the lens to capture hundreds of images with slightly varying points of focus. Depth is recovered by selecting the image plane location that gave the best focussed image (this is similar to how autofocus works in commercial

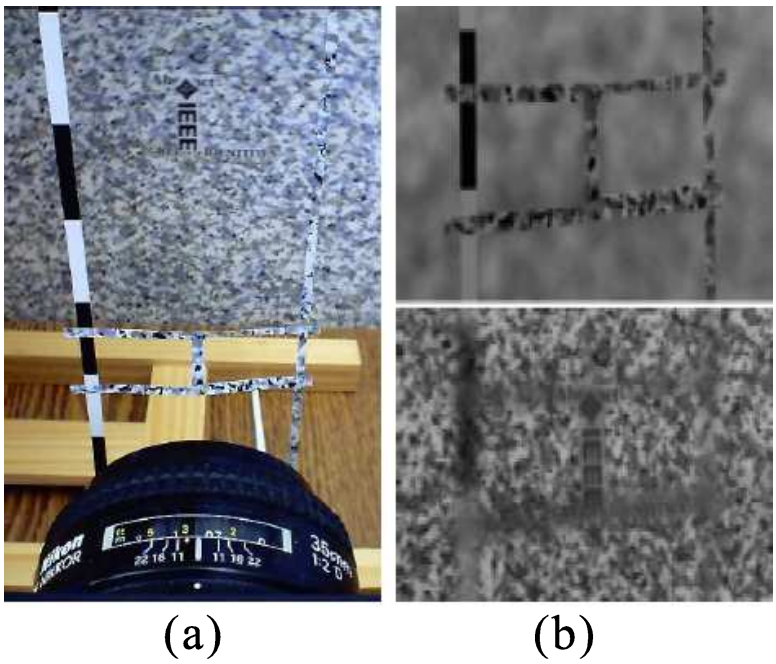


Figure 5.7: Multi-focus capture setup and input by Favaro and Soatto [31].

cameras). This method is limited to static scenes and requires very high frequencies everywhere. In contrast, our optimizer is driven towards a correct solution even in low-frequency areas by the regularization terms.

For defocus experiments, it is common to use small objects. Small objects are good because the aperture is large compared to them, so the PSFs are large and the defocus effect is magnified. Chapter 8 discusses techniques that extract information from defocussed images of human-sized objects where the PSFs are relatively small.

Chapter 8 describes a new algorithm that extends Favaro and Soatto's ideas to solve the matting problem. The major challenges are working with small PSFs and large objects, recovering a matte with accurate fractional values, and solving the problem efficiently in both space and time.

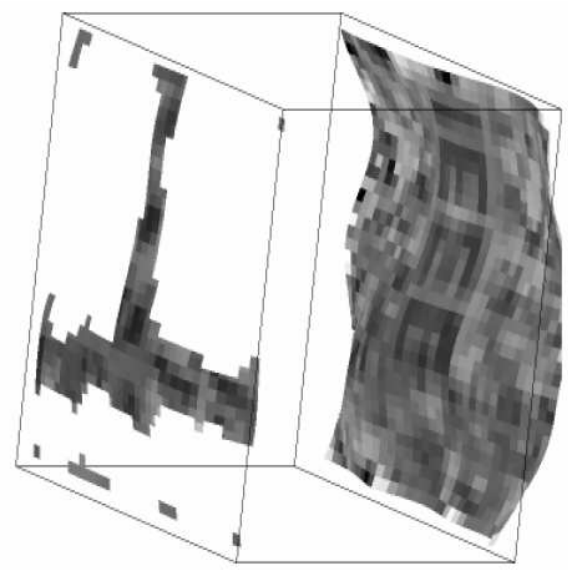


Figure 5.8: Scene reconstructed from defocus by Favaro and Soatto [31].

Chapter 6

Optical Splitting Trees

Beam splitting is a widely used method for creating multiple geometrically similar but radiometrically controlled images of a scene. However, acquiring a large number of such images is known to be a hard problem. This chapter introduces optical splitting trees that can recursively split a monocular view of a scene a large number of times. In this tree, the internal nodes are optical elements like beam splitters, filters, and lenses, and the leaves are video sensors. Varying the optical elements allows one to capture at each virtual pixel multiple samples that vary not only in wavelength but also in other sampling parameters like focus, aperture, polarization, exposure, subpixel position, and frame time. I begin with a motivation of splitting trees, present a diagram system for describing them, discuss desirable properties of a tree, and then show several interesting trees.

This chapter concludes with a discussion of the physical layout necessary to implement an optical splitting tree. I offer a good layout for eight cameras with up to 15 filters inside the tree. This may be the optimal layout; I prove that it is impossible to implement a splitting tree with an arbitrarily larger number of nodes in a finite number of dimensions (e.g., 2D or 3D) if the components are restricted to integer grid points.

6.1 Multiple Views

In the context of computer vision, multiple images that are geometrically similar but radiometrically controlled are useful for many applications. These include such as high dynamic range imaging, focus/defocus analysis, multispectral imaging, high speed videography, and high resolution imaging.

Beam splitting is widely used to create multiple reduced-amplitude copies of the plenoptic field, so that similar sets of light rays may be captured by different imaging systems at the same time. This allows different images of the same scene from the same viewpoint, the goal of the SAMPL camera. I informally refer to this process as “creating multiple copies of an image” and “creating multiple views.”

In order to create multiple copies of an image, one must ensure that the optical paths to the sensors are geometrically similar. All paths must have the same *optical length*, so that the image at each sensor has the same size. The optical axis must enter the sensor perpendicular to the imager. Prior to introducing filters, each path should have the same transport characteristics, i.e., the path itself should not produce filtering unless that is explicitly desired.

Creating the geometrically similar paths is a challenge because it requires precisely locating all elements, which have six degrees of orientation freedom and are each subject to manufacturing abberations that distort the plenoptic function away from the ideal. Creating multiple copies of an image has been demonstrated in various contexts that were reviewed in chapter 5. Rarely have more than three copies of an image been created. This chapter develops a scheme for expressing and building systems that contain more than three views. I have used this scheme, the splitting tree, to capture as many as eight simultaneous views in practice. With $640 \times 480 \times 30\,fps$ sensors, this allows capture of a multi-parameter video with 640×480 pixels $\times 3$ colors $\times 8$ arbitrarily different parameter settings per frame.

6.2 Light Amplitude is Precision

To generalize the notion of making multiple copies of a view, I introduce the concept of an *optical splitting tree*. The splitting tree is any set of optical elements that can recursively split a monocular view of a scene a large number of times without excessive loss of light. Light is a resource, so my philosophy is to direct light that cannot be measured at one sensor to another sensor instead of discarding it. This contrasts with previous approaches to, say, HDR, that use neutral-density filters to intentionally reflect light away from the sensor.

Consider the motivation for capturing multiple copies of an image–it is to later to combine them

in interesting ways. The combined image has *high precision* because it combines the precision of its inputs. However, the combined image cannot have more information than the original incipient light field, since that is what was sampled. From an information theory point of view, every split directs some information down the left branch of the tree and some down the right (and, unfortunately, some information is lost at a non-ideal splitter). The splitting tree is therefore not creating information but partitioning the available information according to a predetermined measurement plan. The splitting tree implements the measurement plan, allocating measurement precision proportional to the amount of light transported along each path. In a physical implementation of a splitting tree it is easy to observe when a measurement is imprecise because it appears noisy (for example, compare the relatively low-light pinhole images in to the large aperture images chapter **??**). Under the photon model of light it is obvious why light amplitude is equivalent to precision–more light means more photons per pixel, which means smaller variance in the measured value per unit time.

A splitting tree is literally a tree in the graph sense. It has the topology of a tree, and the physical layout of the components in such a system naturally assumes the geometry of a regular tree. More importantly, from an analytic perspective, it can be represented by a filter system that is a tree. The edges of the splitting tree are light paths and the nodes are optical elements. Nodes with a single child represent filters and lenses. Nodes with multiple children are beam splitters. Leaf nodes are image sensors (e.g., CCD arrays). The plenoptic field enters the system at the root. Although the physical path length to each sensor's optical center is identical, the *tree-depth* of a sensor (the number of internal nodes between it and the root) may differ.

6.3 Abstract Representation

Figure 6.1 shows a schematic of a full binary splitting tree, viewed from above. Light from the scene enters the tree at the dotted line near the center. The diagonal red elements represent half-mirrors. Each mirror reflects some light at 90-degrees to the incipient optical axis, absorbs a small amount, and transmits the remaining light along the optical axis. The gray disks represent lenses and apertures. They are immediately before the sensors, which are represented by small cyan rectangles. I designed this layout is designed to pack components into a small form factor without occluding any

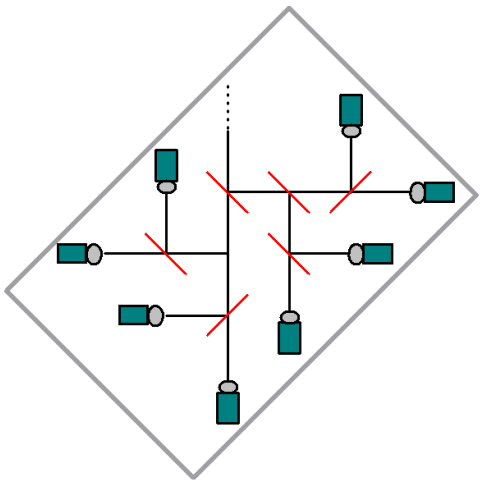


Figure 6.1: Physical layout schematic for a full binary splitting tree with eight sensors.

optical path. The grey rotated rectangle is the boundary of the optical table to which the components are bolted. Later in this chapter I explain why it is rotated relative to the optical axis.

Figure 6.2 (top) shows an abstract representation of same splitting tree. Because all paths from the root to the leaves have the same physical length, the abstract representation need not preserve distances. Angles are an artifact of building the physical system and also need not be represented. What is left is a diagram in which only graph topology is significant. The tree has a branch factor of at most two when the beam splitters are half-mirrors (as in our implementation). Other splitting elements can produce higher degree nodes. When a set of internal nodes serve only to create copies of an image and does not filter the view, we can collapse their representation into a single node with many children, as shown in figure 6.2 (bottom). This representation also abstracts the nature of the of splitting element that is being employed.

6.4 Filters

Creating multiple views is only interesting if the views are captured in different ways. Different images can be captured by slight phase shifts in space and time. Slight translations of the imager perpendicular to the optical axis produce sub-pixel phase shifts. These are useful for capturing super-resolution images, e.g., [9]. Slight translations parallel to the optical axis produce a scale difference in the image and focus at a different depth. The scale change is typically undone in

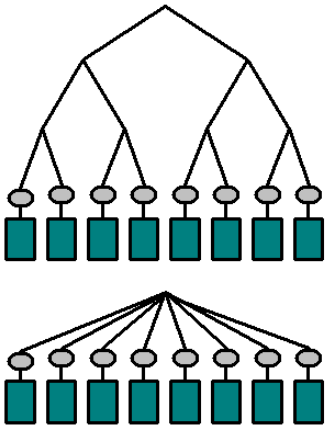


Figure 6.2: Two abstract representations of the full binary optical splitting tree.

software, leaving focus as the only difference. Multi-focus images have applications for scene reconstruction, e.g., [92]. Adjusting the timing offset between cameras produces high speed video, e.g., [41].

Filter and lens components inserted as tree nodes can also differentiate the paths in a tree. Figure 6.3 shows a tree with supplemental internal nodes that are band-pass wavelength filters. Each filter transmits only a narrow range of wavelengths to the monochrome sensor at the leaf below it. This tree produces multispectral video with eight color samples per pixel.

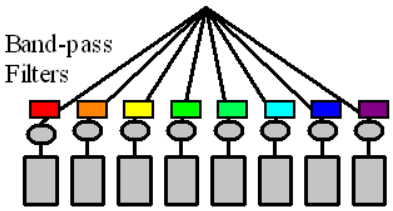


Figure 6.3: Band-pass multispectral splitting tree.

The drawback of using filters in this way is that they discard light. Assume a theoretically ideal splitter with no loss and orthogonal band-pass filters with 100% transmission in their pass band. Even with theoretically ideal components, each sensor receives only $\frac{1}{64}$ of the original light. Even worse, $\frac{7}{8}$ of the incident light is lost (converted to heat) by the system. This is because the eight-way split immediately reduces each path to $\frac{1}{8}$ of the available light (conserving all energy), but the band-pass filters each reflect $\frac{7}{8}$ of the $\frac{1}{8}$ that they receive away from their corresponding sensors.

Figure 6.4 shows an alternative splitting tree for multispectral imaging. This tree also produces eight color samples per pixel. However, with ideal components it has zero light loss and each sensor receives $\frac{1}{8}$ instead of $\frac{1}{64}$ of the original light. The difference from previous tree is that this efficient tree uses dichroic mirrors as splitting elements. Dichroic mirrors divides the spectrum, transmitting half and reflecting the remainder. Stacking them allows creation of the same pass bands as the previous tree, but without light loss. For a given budget, the filter-based tree may still be preferable, however. Manufacturing large dichroic mirrors is expensive and band-pass filters are readily available. Decisions like this are core to the process of designing a multi-view camera. The splitting tree diagrams aid in the process because they strip the irrelevant aspects of camera design and leave what is important–filters, lenses, sensors, splitting elements, and topology.

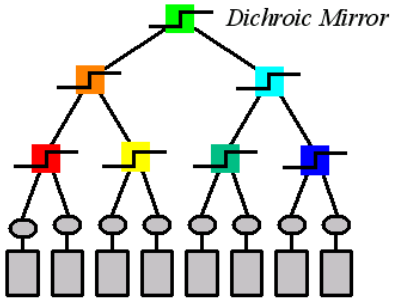


Figure 6.4: Theoretical optimally light preserving multispectral splitting tree.

For some applications like multispectral capture it is desirable to construct these balanced binary trees in which sensors have the same tree depth and the beam splitters divide incident light evenly between their children. In other applications it is useful to unbalance the tree. One may do so either by using beam splitters with an uneven division ratio, or by creating a structurally unbalanced tree where the tree-depths of the sensors vary.

Figure 6.5 shows one such an unbalanced tree. With the exception of the right-most leaf, there are no filters in this tree, and the splitting elements are ordinary half-mirrors. The images produced at each sensor are therefore identical except for amplitude–each sensor receives half the light of its upstream nearest relative. The application for this tree is high-dynamic range video capture. Chapter 9 shows results using this tree with tone mapping software to create range compressed images.

This brings out an important property of the splitting tree. For 50-50 beam-splitters, leaves in the tree with the same depth receive the same fraction of the original of light.

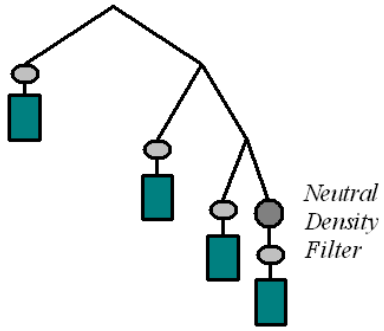


Figure 6.5: Unbalanced splitting tree.

6.5 Physical Layout

The tightly packed physical layout in figure 6.1 is not accidental. It took several design iterations to construct a tree that did not occlude itself (as an exercise, put away the figure and try to draw an eight-leaf tree with equal path lengths and right angle splits; you'll quickly see the challenge!) The first viable layout that I discovered was several times larger than the one in figure 6.1 and very awkward to use because most cameras blocked physical access to their neighbor's calibration screws.

The final layout satisfies several geometric and practical design goals and was arrived at through design and experimentation. The design goals are:

- Pack as many leaves as possible in the smallest footprint
- Place nodes at integer grid points on the optical table
- No node can obstruct another's line of sight
- Nodes occupy slightly more than one grid point of space
- Leaves must allow enough clearance access to calibration screws
- All leaves must be the same optical distance from the root

- All sub-paths meet at right angles

The final design meets these goals. Optical tables are square with a cartesian grid of mounting holes. By placing the optical axes of the system at a 45-degree angle to this grid I was able to pack the components more tightly (1 grid square was too little spacing, but $\sqrt{(2)}/2$ squares was just enough) and use a small optical table because the tree's bounding box is inherently rotated. This layout supports eight leaves and $15 = 8+4+2+1$ internal nodes (since a filter can be placed immediately before the leaf). It is essentially a recursive layout, where each subtree rotates 90-degrees away from its siblings to avoid obstruction.

The layout is good, but it raises the question of whether one can do better. Can the recursive pattern iterate indefinitely, or is there a limit to the number of leaves that can be packed by this scheme?

6.5.1 Proof

Theorem: *It is impossible to build an n-leaf splitting tree on a regular rectangular grid in 2D or 3D for arbitrarily large n.*

Geometric Proof: Consider a fully balanced binary tree with n leaf nodes to be built in 2D. The leaves contain half the nodes of a binary tree, so there are an additional n internal nodes arranged in $\log n$ tiers radiating away from the root (plus or minus; all numbers in this proof suffer some rounding error for specific values of n). Assume a regular rectangular grid like the optical table, so that nodes are located at integer locations, and that each node fully occupies one grid square.

In the layout of this tree, the distance from the root to each leaf must be proportional to the height of the the tree. The distance must also be the same for all leaves, since the optical paths must have uniform length. The grid constraint requires that the root-leaf distance be measured as manhattan distance. The distance from the root to any leaf is therefore $k \log n$ grid units, where k is an integer.

Without loss of generality, assume the root is at the center of the layout. The leaves must lie within a rotated square (diamond) shape about the root dictated by the constant-manhattan distance requirement for the optical paths. The diagonals of this square (which are purely horizontal and

vertical on the grid) have length $2k\log n$ because leaves may lie to either side of the root. The area of the square within which all components must lie is therefore $2k^2(\log n)^2$. A total of $2n$ components are required to build the tree, and each component occupies one unit of space. For any constant k and large values of n, the area is required less than the space available since:

$$k\log n < \sqrt{n} \tag{6.1}$$

$$2k^2(\log n)^2 < 2n. \tag{6.2}$$

It is therefore impossible to build an arbitrarily large tree in 2D with regular rectangular grid spacing between components. The same argument holds for 3D and higher; only the exponent on the log changes.

Note: It may be possible to solve the layout problem in 3D using non-unit spacing. The difficulty with a non-unit scheme is that the spacing between elements needs to fall off quickly, which tends to bunch the components together so that there is no space. Other potential schemes are to allow optical paths to cross one another, as is done in a fiber optic cable, or to allow non-right angles. While it is interesting to speculate on the theory of such unusual designs, unfortunately they seem impractical for a real camera.

Chapter 7

Capture System

Previous chapters described the theory of light, images, and camera design. This chapter marks the transition from theory to applications in the thesis. It gives the concrete details of how to build a reconfigurable splitting tree system that can be quickly adjusted to implement any particular splitting tree.

The reconfigurable capture system hardware comprises:

- 8 color and 8 monochrome sensors
- 8 high-quality objectives (“lenses”)
- IR, visible band pass, and neutral-density filters
- 1 hot-mirror and 10 half-mirrors
- Computer with GeForce FX 6800 graphics and 1 TB of storage
- 24” $\times$ 36” aluminum optical benchplate
- Wheeled cart
- Battery pack for several hours of remote operation
- Custom mounting stages for all components
- Synchronization, data, and power cabling

- Several thousand watts of lighting
- High-power digital projector
- Tripods, color charts, and other photographic apparati

Including labor costs, the total camera cost is about $25,000. Half of that cost is in the filters and lenses (fortunately, many parts were already available in the laboratory at MERL). This is a prototype and an extensive setup for experimenting with camera design; a non-mobile, eight-sensor system with fewer filters could probably be produced for about half the cost.

I used standard components wherever possible. In a few places no existing part met the design specifications at a reasonable price, so I manufactured a new component. This was the case for the mobile workbench, the 6-dof mounting for the sensors, the hardware synchronizer, and of course the calibration and capture software.

7.1 Mobile Workbench

In order to perform a wide range of experiments in and out of the laboratory, the entire camera system is built onto the wheeled cart shown in figure 7.1. The cart is 2 ft wide so that it can pass through standard doorways. The lowest level of the cart supports two battery packs and 100 ft of extension cord to power the system in any circumstance. Above that is a rack holding the PC and a shelf that supports the LCD monitor, mouse, keyboard, and utility tray. The utility tray houses components that are not currently in use. The benchplate is bolted to the top shelf, at eye level.

The smallest optical benchplate that supports the eight-leaf splitting tree layout from the previous chapter is $2ft \times 3ft$ with $1/2$ inch in hole spacing (Edmund Optics part NT03-680). The cart itself weighs about 60 lbs. The batteries and PC in a rugged all-metal case add another 60 lbs. The benchplate weighs 40lbs, and the cart is typically loaded with tripods, lights, and props. The entire setup is about 150 lbs. However, large pneumatic tires enable one person to move the cart and the weight is an advantage when filming because it stabilizes the platform. When filming outside walking distance from the laboratory it is necessary to disassemble the system and transport the pieces in a cargo van.

Figure 7.1: The mobile optical workbench with the splitting tree system on top, at eye level.

A custom foam case (not shown in the figure) fits over the benchplate to protect the system and block stray light. Because the cameras are sensitive in the IR range as well as the visible range, the case is painted black on the inside and covered with aluminum foil on the outside. An unfortunate side effect of the aluminum covering is that the case appears amateur despite being extremely functional.

7.2 Mounting Stages

The optical benchplate provides a high degree of precision in translation, however the sensors themselves have very loose manufacturing tolerance along all three translation axes and all three rotation axes and do not align properly if simply bolted to the table. To overcome this deficiency I attach each sensor to a 6-degrees-of-freedom mount and physically calibrate the system using video feedback. The mount has several design restrictions:

1. Short (small y-dimension) for stability
2. Small footprint (small $x-$ and $z-$ dimensions) for filter size
3. Low mass to reduce resistance
4. Inexpensive
5. Precise for micro calibration
6. Stable to maintain calibration
7. Independent axis control.

It is desirable that the mount be short because placement precision is controlled by the table. The higher that components rise above the table, the longer the lever arms on which they pivot and the less placement precision available. Also, high placement leads to more susceptibility to vibration and miscalibration while shooting. It is desirable that the mount be small in other dimensions so that the sensors can be packed close together. Close sensors mean a shorter path from the common aperture to the sensors, which allows the beam splitters and filters to intersect the view cone at a

smaller cross-section. Beam-splitters and filters have cost proportional to area, so a small mount allows more affordable optical components.

The mount must have low mass so that higher stages do not place extraordinary stress on lower stages. Ideally, the mass of the mount should be negligible compared to the mass of the sensor. Since most possible designs involve either tensions or springs to resist adjustment screws, low mass reduces the resistance strength required. All parts on the camera must be replicated at least eight times, so expense is a constant constraint even for the prototype system.

The mount must be precise in order to allow adjustment on the order of tenths of a millimeter and tenths of a degree because the CCD pixels are on approximately that scale. It must be able to maintain its position stably so that the system does not need to be recalibrated too frequently. The calibration system requires the ability to adjust each axis with a fair level of independence so that error along each dimension can be reduced in sequence.

I investigated commercially available 6-dof stages. All were expensive and lacked stability for the mass of our sensors (200g with an objective). Most of the commercial designs were focussed on providing a larger range of motion but lower degree of stability compared to the requirements for this project. The screw based translation mounts lacked convenient mounting mechanisms and rotation systems. Ball-and-socket designs lacked independent axis control.

MERL engineers William Yerazunis, consultant John Barnwell and I collaborated to design a comparatively inexpensive mount system satisfying the design constraints. We rejected several designs involving ball bearings that introduced too much instability and imprecision and ones with external springs that had large footprints before arriving at the following design. The detailed engineering specifications for the parts are given in appendix B.

Figure 7.2 shows a complete sensor mount in false color, with a Basler video camera for reference. The mount contains three stages. Each stage allows translational adjustment of ± 30 mm and rotational adjustment of $\pm 10^\circ$ via thumbscrews (gold). The stage comprises a base and two moving parts, the slide and T-plate, constructed of aluminum and two screws and springs. The light yellow L-brackets provide 90° rotations between axes. The entire bracket bolts to the optical benchplate at the cyan L-brackets.

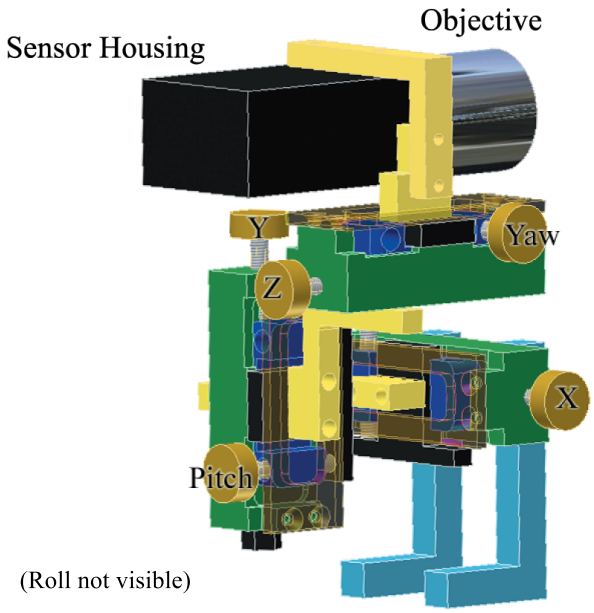


Figure 7.2: Sensor mount comprising three stages, seen from behind.

The *base* (figure B.1) is a rectangular slab with a U-gouge that runs the length and a hole straight through the base at one end of the U-gouge. The *slide* (figure B.2) rides in the U-gouge and provides translational motion. It is a slab the width of the U-gouge with a cube tab that hangs down into the hole at the end of the U-gouge, which is the width of the tab but longer along the translation axis. The translation thumbscrew presses against one side of the tab and a strong spring opposes it. The design is intentionally asymmetric: tab and the hole that it sits in are far to one side of the translation axis. This is because the translation spring has a minimum length when compressed that must be concealed within the body of the base to avoid interfering with other stages.

The *T-plate* is bolted to the slide and may rotate freely about the bolt. The arms of the T project slightly past one side of the short axis of the base. The rotation thumbscrew and opposing spring mount through holes on the top of the slide and press against opposing arms of the T. The rotation spring must be much shorter than the translation spring because the base is narrower than it is long. This leaves the mount with less stability on the rotation axes and slight pressure can compress the rotation spring, an undesirable quality. It is therefore crucial that the pitch stage is oriented so that the weight of the sensor is opposed by the screw and not the spring.

The entire mount is held to the table by a clamp consisting of a large plate and a bolt. This allows gross translational alignment of the entire mount in *xz* and rotation about *y* by sliding the mount relative to the bolt-hole before tightening. The space between the two feet of the mount is greater than $\frac{1}{2}$ inch, so that every possible translation over the surface of the optical table can be achieved by sliding relative to the bolt hole or moving to the next adjacent hole.

7.3 Beam-Splitters and Filters

A beam-splitter is thin slab of glass that mirror reflects about 50% of incident light and transmits about 50% to be transmitted. As previously discussed, manufactured beam-splitters are not ideal. There is some light loss to absorbtion and diffuse reflection, and the mirror reflection and transmission ratios are not perfectly uniform over frequency and polarization. The beam-splitters are rated for collimated, coherent light and not natural light. They are not perfectly flat, so minor distortion of the viewed image can occur. In total, for a real system we can expect about 10% light loss on each path and the reflected light to be slightly polarized. Image distortions are generally smaller than a pixel; precise placement and orientation of the beam-splitter is a larger problem than imprecision in the shape. Square beam-splitters in 25mm increments are standard optical equipment and are readily available. The cost of manufacturing a beam-splitter is roughly linear in the surface area, although splitters larger than 200mm on a side are rare and may incur special order costs.

Placing a beam-splitter on the optical path for an image sensor at 45-degree angle allows a second image sensor to observe the same path (albeit left-to-right inverted). The two sensors then share a virtual center of projection. The configuration is symmetric, so each sensor will image the sum of two views. The first is along the original optical path, and the second is a garbage view of somewhere inside the SAMPL camera, at a right angle to optical axis. I place a matte black card in that second view. The net image captured is thus the view around the original path, albeit with only 50% of the original intensity.

For a splitting tree of depth three, the largest beam splitter (at the root) must be about $100 \times 100mm^2$ and the smallest (close to the leaves) $75 \times 75mm^2$. Splitters any larger cannot pack closely and are expensive; smaller splitters would restrict the field of view excessively.

The hot mirror is a special beam-splitter that transmits visible light and reflects IR. I use a 100mm square hot mirror (a larger variation of Edmund Optics part NT46-388) for multi-modal capture of IR and visible light.

Neutral density filters block light at a uniform rate across the visible spectrum. They are used to match the amplitude of images captured with the same exposure time but different apertures (as is done in the next chapter). The neutral density filters (Edmund Optics part NT54-737) used on the SAMPL camera have an optical density of 0.6. *Optical density* is the negative base-10 logarithm of the transmission ratio, so this is a $10^{-0.6} \approx 25\%$ transmissive filter. Less transmissive filters are created by gluing multiple filters together.

7.4 Sensors

The video sensors are Basler a601fc Bayer filter color cameras and Basler a601f monochrome cameras with 640×480 resolution at 30 fps. The specifications for these cameras are those given at the end of chapter 3. The cameras connect to the computer through the IEEE 1394 (FireWire) bus. PCI adapter cards for the computer accept three FireWire devices each. The computer is equipped with three of these, providing one port for each of the eight sensors and a final port for the external Lacie hard drive on which recorded video is stored.

7.4.1 Objectives

Each camera is equipped with a $50mm$ C-mount objective for a narrow field of view that matches the 100mm square root beam-splitter. Cosmicar and Pentax both produce high quality objectives; the Cosmicar one was chosen because it had the larger aperture ($f/1.4$). Radial distortion and chromatic abberation are negligible for these lenses.

7.4.2 Synchronization

The Basler cameras are synchronized through an external hardware trigger cable. Although commercial triggering devices are available, I chose to build a hardware interface that allowed the computer to directly control the hardware trigger. This was significantly less expensive (commercial

trigger devices cost about $3000 and my solution contains about $20 worth of parts and 4 hours of labor). It also allows independent computer control of the trigger signal for each sensor, which is necessary for high-speed video capture.

The hardware interface is a cable that connects each of the eight data pins (numbered 2 - 9) of the PC's parallel port to the trigger pin (number 5 on the 10-pin RJ-45 port) of a different sensor. The ground wire (pins number 18 - 25 on the parallel port, pin 6 on the camera's RJ-45 port) is shared between all sensors and the parallel port. Using a custom device driver[1], the computer sends bit patterns directly to the sensor through the parallel port. Because each sensor corresponds to one bit, it can either trigger cameras independently, for example, to create a high-speed camera, or simultaneously in order to provide synchronized video. Between trigger signals the parallel port data bits must return to zero so that subsequent triggers can be distinguished.

The synchronizer is based on one Matusik built for the 3DTV system [53]; the variation used on the SAMPL camera was so successful that a later version of the 3DTV demonstrated at SIGGRAPH 2005 incorporated my modifications.

The Basler cameras are designed to wait 1/30th of a second between frames, and then capture a frame on the rising edge of the next square wave. This means that unless the triggering square wave is precisely 30 Hz and synchronized with cameras, the frame rate may drop as low as 15 fps (a similar problem occurs in rendering if one waits for the video refresh before updating the display). I clock the trigger signal at 1000 Hz, which allows the cameras to asymptotically approach the ideal 30fps capture rate. At this $30 - \varepsilon$ fps rate, the cameras may have slightly longer than half the period of 60 Hz fluorescent lights, producing the impression of pulsing lighting in the recorded video. I therefore illuminate scenes with halogen bulbs or daylight.

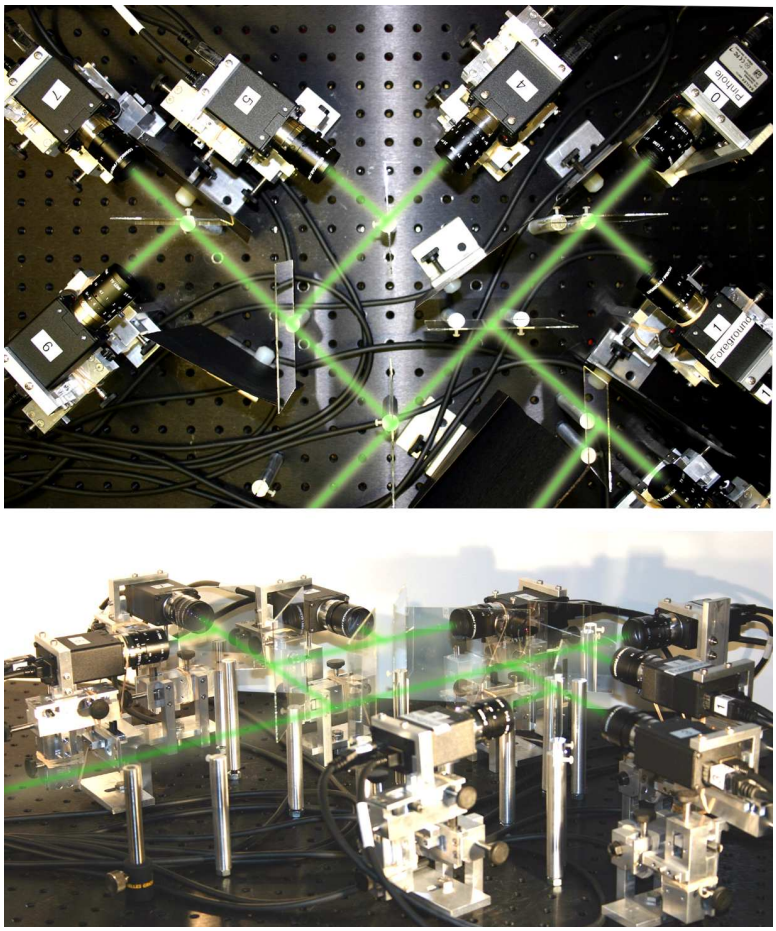


Figure 7.3: Optical components implementing a full binary optical splitting tree.

7.5 Configurations

7.5.1 Full Tree

The optical path is visualized using a superimposed green beam. In the side view, the black backing cards have been removed from the beam splitters to give a clearer view of the system.

The trees used in other experiments (except for multi-axis) are all subsets of this tree.

7.5.2 Matting

Figure 7.4 shows the configuration used by the matting application described in the next chapter. The figure is oriented so that light from the scene enters on the right. The red line is a visualization of the optical axis. It forks at the beam-splitter, which has been false-colored light blue for the figure. The transmissive path leads to a pinhole video camera on the far left. The reflective path hits a second beam-splitter and then passes through two neutral density filters (shown in yellow) along each branch. Two sets of two filters are used because the filters are not large enough to cover the entire field of view in front of the second beam-splitter. This is an important property of the splitting tree–the filtering effects of the tree as a whole is unchanged if any filtering node is duplicated and then pushed down into its child branches. This is equivalent to distribution in algebra; in practice, passive optical filters are also associative and commutative, so filters may be freely moved through the tree for implementation reasons so long as the total set of nodes along every path to a leaf remains unchanged.

The camera on the lower right is focussed on the background of the scene; the camera in the center is focussed on an actor. The black backing cards are visible behind the beam splitters in the photograph.

7.5.3 Multi-Axis

This thesis describes single-axis video. Multi-axis arrays are also useful. Figure 7.5 shows an eight-camera array I designed that is driven by the same capture system, creating a multi-axis, or

[1]There are many commercial and freeware drivers that allow direct access to ports under Microsoft Windows; inpout32.dll is the freeware solution I use.

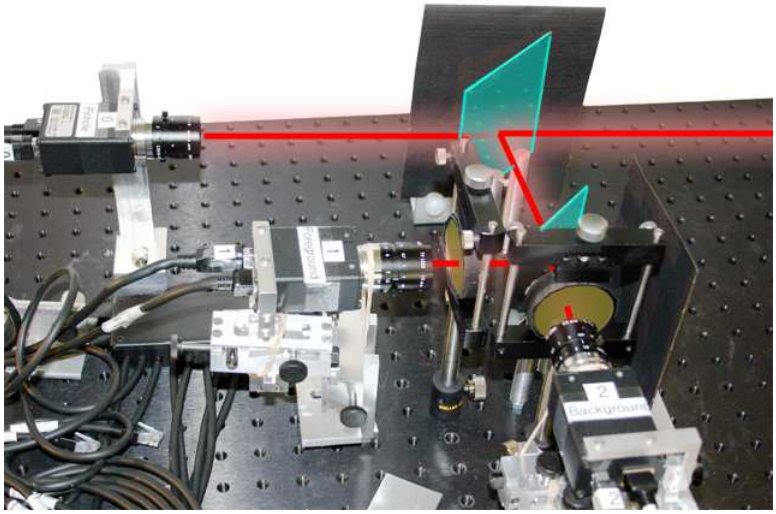


Figure 7.4: Pinhole, foreground-, and background-focussed camera for defocus matting.

"MAMPL," camera. Applications for the array camera are not discussed further in this document. See Vetro et al. [85] for the results captured with this configuration.

7.6 Computer

The system is driven by a single 3 GHz Pentium 4 PC running the Windows XP operating system. A small LCD screen connected to the PC outputs the individual video streams for rapid feedback during calibration and video capture (e.g., see figure 7.6).

The computer has 2 GB of RAM and 1 TB of disk. All data is recorded to pre-allocated buffers in memory and then dumped to disk. For Bayer eight streams at full resolution the camera can capture about 15 seconds of video before exhausting main memory. Zitnick et al. report that on their system they are able to record eight high resolution streams directly to disk [95]. It should be possible to achieve the same bandwidth with the SAMPL camera given more software infrastructure and a faster disk.

7.6.1 Calibration

The difficulty of calibrating an optical system naturally increases with the number of elements. Cameras that share an optical center are also more difficult to calibrate than systems using stereo

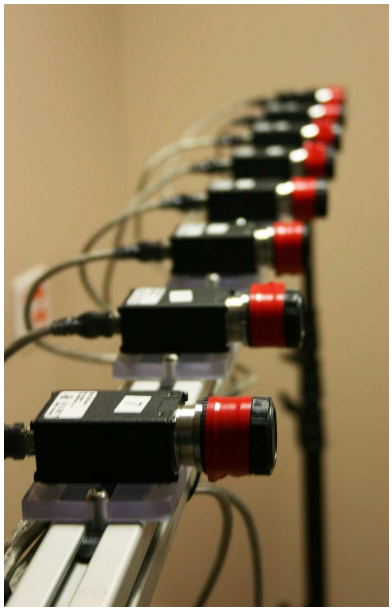


Figure 7.5: Eight camera array for multi-axis multi-parameter video.

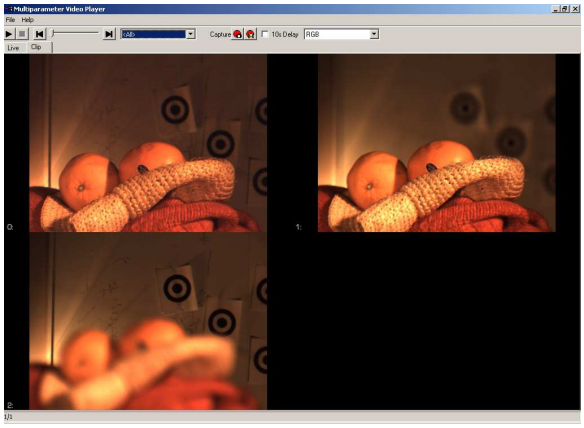


Figure 7.6: Screenshot of the Multi-parameter Video Player software.

pairs where the images are not expected to align. I calibrate our system in three stages: First, I align the beam splitters; second, I align the cameras; and third, I compute a homography to correct the remaining mis-registration in software.

Rotations of cameras relative to the optical path can be perfectly corrected (up to the sampling precision) by the homography. The optical paths within the system are generally short compared to depths in the scene, so the position of cameras along the optical path (say, within the tolerances provided by an optical table) is not critical. So the primary concern for calibration of the physical components is translation of the sensors perpendicular to the optical axis, which produces parallax that cannot be corrected in software.

I use half-mirrors for beam splitting, which must be rotated at 45-degrees to the optical axis. To orient these, I place a lens cap over each camera and shine a laser through the first beam splitter. This produces a single dot near the center of each lens cap. Working through the splitting tree from the root to the leaves, I rotate the beam splitters until each dot appears in the center of the lens cap.

I then construct a scene containing a target image (five bulls eyes) printed on transparent plastic as a foreground element and a background element of the enlarged target printed on poster board. I move the foreground target until its pattern exactly overlaps the background target in the view of camera 0. I then translate all other cameras until the target patterns also overlap in their views, rotating the cameras as needed. Because our camera mounts do not rotate around the camera's optical center, the rotation can introduce a new translation error, making it necessary to repeat this process several times.

Finally, I compute a homography matrix for each camera to map its view to the view of camera 0. The matrix is computed from corresponding points that are either chosen manually or automatically by imaging the movement of a small LED light throughout the scene. The automatic method is convenient in cases where it is hard to visually choose corresponding points, such as when the cameras are focused at different depths or receive different amounts of light for HDR capture. I compute an affine matrix by solving the standard least squares problem given the corresponding points. Although I have not done so, it is also possible to compute an arbitrary deformation from the corresponding points to account for lens abberation.

The calibration process takes about four hours for a single human operator. Most of the time is spent adjusting the orientation of components on the optical table using wrenches and screwdrivers. Fortunately, components can be locked down once a large splitting tree is built and calibrated. Changing filters, focusing, adjusting apertures, and many other adjustments do not affect the branch structure of the tree. Recomputing the homography takes about one minute.

I calibrate the physical cameras to within a few pixels and then correct remaining error in software. The process is repeated after relocation because our camera mounts are easily disturbed.

The primary challenge is aligning the optical axes so that there is no parallax between cameras. I place a large white poster of a quincunx pattern of five black bull's eyes 8m from the camera. I place a transparent sheet printed with a small version of the pattern 2m from the camera so that it precisely occludes the distant pattern in one camera's view. I translate the remaining cameras until the distant pattern is overlapped by the near one in each view. As with the sights on a rifle, the alignment of points on two planes under projection guarantees no parallax.

Focusing a camera also changes the size of the image produced, so even perfectly aligned cameras produce differing images. I correct for this with an affine transformation in software. I have used two methods with equal success. Before capturing data I shoot a scene containing a bright LED moving perpendicular to the optical axis at the foreground camera's midfield depth. The LED's centroid is easy to track (even when defocused) and provides a set of points from which I solve for the least-squares transformation (called a homography matrix). I repeat the process for the far plane. When it is inconvenient to use the LED (e.g., the far plane is in the center of a busy street), I can also manually select feature points in a single set of three still frames after capture. Note that the calibration process is performed only when the camera has been physically disrupted—I do not calibrate each video sequence.

I color correct the images by solving a similar problem in color space. Here, the feature points are the colors of an image of a color chart (figure 7.7 and the affine transformation is a color matrix.

I apply color and position correction in real-time to all streams in OpenGL on a GeForceFX 6800 GPU for both preview and capture. Each video frame is loaded into a rectangular texture and rendered with the homography as the texture-coordinate transformation matrix and a fragment

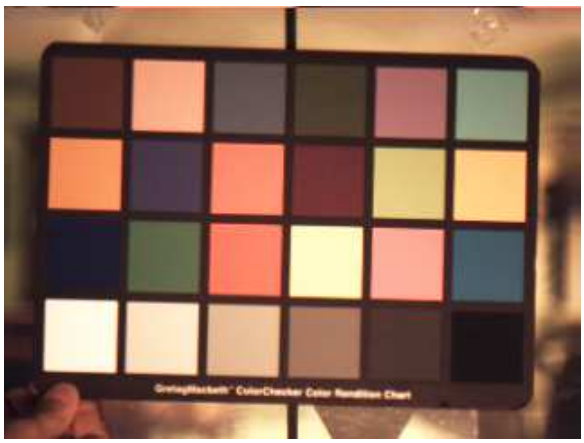


Figure 7.7: Color calibration chart.

shader that multiplies each texel value by the color matrix.

Chapter 8

Defocus Matting

This is the first of two chapters that discuss applications of the system, theory, and framework presented previously in this thesis. Applications solve a problem, and one does not need to look hard to find new problems in graphics and vision. Many of the classic problems that began these fields have never been solved in a sufficiently general context. Thus most good new problems are really old problems addressed without the simplifications that were introduced in the past.

This chapter tackles the classic matting problem of separating an image into foreground and background elements. This is one of the oldest in computer graphics, which was first investigated in the 1950's. The practical solution to the matting problem is well-known to every film aficionado: the blue-screen. More recent solutions use machine learning and gradient analysis techniques. These compute mattes for natural images filmed without blue screens, but they require a user to edit most of the frames and correct the final results.

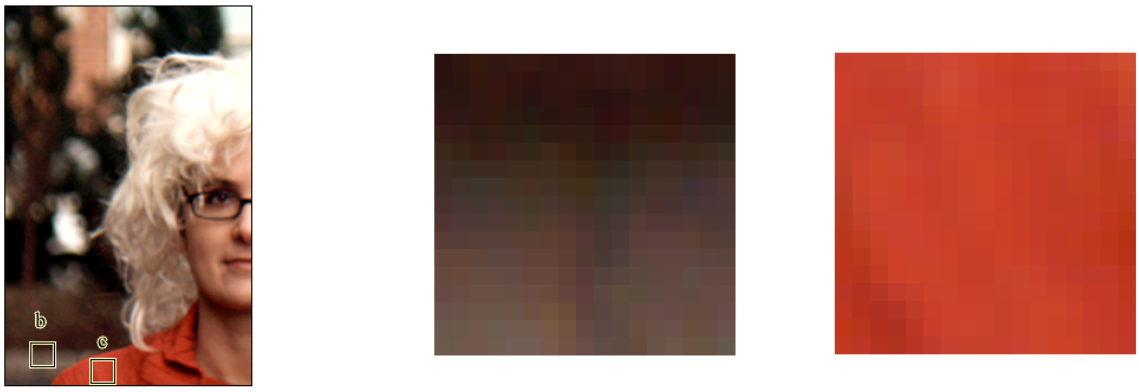
There are three applications in this chapter: a practical solution to the static background matting problem called Defocus Difference Matting, a method for automatically computing trimaps from multi-parameter video, and a more ambitious general matting algorithm called Defocus Video Matting. The latter application uses ideas from the first two. It solves the classic matting problem in the general case for the first time. Defocus Video Matting operates on video of fully dynamic scenes, without the artificial constraint of a static or blue screen, and it requires no user interaction. The key idea behind all three applications is to separate foreground and background using defocus information encoded in multi-parameter video.

Although it has not been previously used in the matting literature, using focus to discriminate foreground and background is logical and intuitive. The foreground is the focussed area of an image and the background is the defocussed area. However, making an accurate depth estimate for objects based on defocus is an open hard problem in computer vision with a large body of literature [12, 6, 5, 83, 92, 39, 69, 46, 29, 24, 81, 17, 62].

The object-depth from defocus problem is hard despite the seemingly intuitive process for solving it because the human visual system is very good at identifying defocussed objects but it does so in a subjective high-level manner that depends on common sense reasoning that is not easy to reproduce with an algorithm. Figure 8.1 demonstrates the difficulty. The goal is to classify the pixels at the centers of the two marked boxes in (a) as focussed or defoccused. In the full image (a), it is obvious to a human observer that the actor in the red sweater is in sharp focus and the background is defocussed. Parts (b) and (c) of the figure show the square subregions under high magnification. This is how an image processing system views the scene. Examining only these zoomed-in pixels, it is no longer obvious that the center of region (b) is defocussed and the center of region (c) is in focus. The hue and intensity gradients are small in each image, and none of the sharp edges that people associate with in-focus images are visible in either. The red sweater is simply too uniform and fuzzy of an object to appear sharply focussed. The subregions shown have hundreds of pixels, which makes them very large compared to the neighborhoods that most image processing filters use as kernels. A typical gradient discriminator has a 5×5 or smaller kernel and has no chance of correctly classifying these images. To solve the matting problem efficiently, it is necessary to not only determine that the red sweater pixel at the center of (c) is in focus with a real-time (small kernel) filter, but to make that determination at a sub-pixel resolution *and* to produce results that are coherent over a sequence of video frames so that the edges of objects do not shimmer.

To solve this problem, I begin with an easier sub-problem of pulling mattes from images with known, high-frequency backgrounds. The defocus difference matting algorithm in this chapter solves this sub-problem and offers many of the practical advantages of blue-screen matting without some of the disadvantages.

Defocus difference matting is based purely on the algebra of image formation. To solve the truly



(a) (b) (c)

Figure 8.1: Closeup demonstrating the difficulty of determining focus at the pixel level.

general matting problem, the ultimate *defocus video matting* technique presented in this chapter must go beyond the pure derivation to include terms based on the likelihood of a given solution. This is analogous to how the human visual system probably solves the quandary of Figure 8.1: the observed area of the red sweater is *likely* in focus even though it has no sharp features because other connected parts of the image have the same hue and contain sharp edges. That is, adjacent and similarly-colored pixels are likely from the same object and must therefore all be in or out of focus together. In defocus video matting, "common sense" heuristics like this join expressions derived directly from the image formation equations. The heart of the algorithm is a non-linear optimizer for a huge sparse system of equations that seeks the a likely matte for an video sequence.

This chapter uses the notation introduced in chapters 5, 4, and 6. Table 8.1 is a reference for the variables used in this chapter.

8.1 The Matting Problem

Previous work on the matting problem was reviewed extensively in chapter 5. A brief summary follows in the next two sections to refresh the salient points for this chapter.

Video matting is the process of pulling a high-quality alpha matte and foreground from a video sequence. Current techniques require either a known background (e.g., a blue screen) or extensive user interaction (e.g., to specify known foreground and background elements). The matting problem

$\alpha[\cdot]$	Matte image for the foreground plane
δ	Impulse function or small step, depending on context
$B[\cdot]$	Image of the background plane
β	Manually-tuned regularization coefficient
$C(\cdot)[\cdot]$	Image composition equation mimicking I
$\vec{E}(\cdot)$	Error vector function to be minimized (eq. 8.12)
ε	Regularization term or small value, depending on context
ϕ, γ	Regularization terms
$F[\cdot]$	Image of the foreground plane
$g[\cdot], h[\cdot]$	Point spread functions (eq. 8.25)
$I_P[\cdot]$	Pinhole image (eq. 8.1)
$I_F[\cdot]$	Foreground-focussed image
$I_B[\cdot]$	Background-focussed image
$I[\cdot]$	Multi-parameter image combining I_F, I_P, and I_B
$\mathbf{J}$	Matrix of error partial derivatives (eq. 8.19)
K	Number of pixels in the input image
N	Number of pixels in the output matte
$Q(\cdot)$	Error function to be minimized (eq. 8.13)
∇	Gradient operator
$\otimes$	Convolution operator
u	Reconstructed scene; unknown in the optimization (eq. 8.11)
Ω	Unknown region of trimap (eq. 8.8)

Table 8.1: Symbols used in defocus matting.

is generally under-constrained, since not enough information has been collected at capture time. I describe a novel, fully autonomous method for pulling a matte using multiple synchronized video streams that share a point of view but differ in their plane of focus. The solution is obtained by directly minimizing the error in filter-based image formation equations, which are over-constrained by a rich multi-parameter data stream. This new system solves the fully dynamic video matting problem without user assistance: both the foreground and background may be high frequency and have dynamic content, the foreground may resemble the background, and the scene is lit by natural (as opposed to polarized or collimated) illumination.

Matting and compositing are some of the most important operations in image editing, 3D photography, and film production. Matting or "pulling a matte" refers to separating a foreground element from an image by estimating a color F and opacity α for each foreground pixel. Compositing

is used to blend the extracted foreground element into a new scene. α measures the coverage of the foreground object at each pixel, due to either partial spatial coverage or partial temporal coverage (motion blur). The set of all α values is called the alpha matte or the alpha channel.

Because of its importance, the history of matting is long and colorful [82]. The original matting approaches require a background with known, constant color, which is referred to as *blue screen matting*, even though green is preferred when shooting with digital cameras. Blue screen matting has been perfected for half a century and is still the predominant technique in the film industry. However, it is rarely available to home users, and even production houses would prefer a lower-cost and less intrusive alternative. On the other end of the spectrum, rotoscoping [**?**] permits non-intrusive matting but involves painstaking manual labor to draw the matte boundary on many frames.

Ideally, one would like to pull a high-quality matte from an image or video with an arbitrary (unknown) background, a process known as *natural image matting*. Recently there has been substantial progress in this area [80, 43, 20, 19, 84]. Unfortunately, all of these methods require substantial manual intervention, which becomes prohibitive for long video sequences and for non-professional users.

The difficulty arises because matting from a single image is fundamentally under-constrained [82]. The matting problem considers the input image as the *composite* of a foreground layer F and a background layer B, combined using linear blending [73] of radiance values for a pinhole camera:

$$I_P[x,y] = \alpha F + (1-\alpha)B, \tag{8.1}$$

where αF is the (pre-multiplied) image of the foreground element against a black background, and B is the image of the (opaque) background in the absence of the foreground. Matting is the inverse problem with seven unknowns ($\alpha, F_r, F_g, F_b, B_r, B_g, B_b$) but only three constraints (I_{Pr}, I_{Pg}, I_{Pb}). Note that blue screen matting is easier to solve because the background color B is known.

This chapter advocates a *data-rich imaging* approach to video matting. It introduces a novel imaging setup that records additional information during capture, thereby constraining the original ill-posed problem. This preserves the flexibility of non-intrusive techniques since only the imaging device is modified, not the scene, while offering full automation. The additional information comes from *defocus*: three pixel-aligned video streams are recorded with different focusing distance and

depth of field.

8.2 Related Work

Most natural image matting approaches [20, 43, 19, 79] require user-defined trimaps to compute the color distributions of F and B in known regions. Using these distributions they estimate the most likely values of F and B for the unknown pixels and use them to solve the matting equation (8.1). Bayesian matting [20] and its extension to video [19] arguably produce the best results in many cases. But user-painted trimaps are needed at keyframes; this becomes tedious for long video sequences. We propose a solution that operates without user assistance and that can be applied as a batch process for long video clips today and will eventually run in real-time on live video as it is recorded.

In addition, robust estimation of color distributions works only if F and B are sufficiently different in the neighborhood of an unknown pixel. The new techniques in this thesis can pull a matte where foreground and background have similar color distributions if there exists sufficient texture to distinguish them in defocused images. In cases where there are neither high frequencies nor color differences, the natural image matting problem is insoluble. Where the defocus problem is ill-conditioned, defocus matting introduces regularization terms inspired by previous matting algorithms to guide the solver to a physically probable solution.

Poisson matting [84] solves a Poisson equation for the matte by assuming that the foreground and background are slowly varying compared to the matte. Their algorithm interacts closely with the user by beginning from a hand-painted trimap and offering painting tools to correct errors in the matte. Defocus matting works best with high-frequency backgrounds, so it complements Bayesian and Poisson matting, which are intended for low-frequency backgrounds.

The basic strategy of acquiring more pixel-aligned image data has been successfully used in other computer graphics and computer vision applications, such as high-dynamic range [27, 61], confocal [50], super-resolution [9], depth estimation from defocus [62, 92], depth estimation from focus [70, 5, 92, 64]. The focus-based depth estimation techniques inspired our approach. However, they are not directly applicable to the natural video matting problem. First, reconstruction

techniques cannot produce fractional alpha values, so they are limited to opaque super-pixel structures. Second, depth-from-focus requires hundreds of defocussed images for a single frame so it is only appropriate for stills. Third, depth-from-defocus is only reliable with active illumination, and for natural matting a passive technique that does not interfere with the scene is desirable. Infrared illumination avoids visible patterns but does not work many scenes. This is because like radar, active techniques work best with diffusely reflective surfaces and can give poor results on mirror reflective or absorptive (black) surfaces that do not reflect the active illumination towards the camera, or in the presence of IR interference, e.g., from direct sunlight. These drawbacks apply to active IR systems like the Zcam [93].

The closest work to defocus matting is a scene reconstruction method by Favaro and Soatto [31]. Both methods use defocussed images and both use gradient descent minimization of sum-squared error, a common framework in both graphics and vision. They solved for coarse depth and binary alpha; defocus matting solves for alpha only but achieve sub-pixel results and an orders of magnitude speedup (precisely, O(image size)) by using exact differentiation. I work with color video instead of monochrome still images, which necessitates a new capture system and calibration. Color is needed to over-constrain the problem and video to reconstruct partly occluded background pixels. I also extend Favaro and Soatto's regularization terms; because matting equations can be ill-conditioned at some pixels, finding good regularization terms continues to be an active area of research, e.g., [4, 13].

Zitnick et al. [95] were the first to create a passive, unassisted natural video matting system. They capture video on a horizontal row of eight sensors spaced over about two meters. They compute depth from stereo disparity using sophisticated region processing, and then construct a trimap from depth discrepancies. The actual matting is computed by the Bayesian matting [20] algorithm on a single view; Zitnick et al.'s contributions are the physical system and stereo trimap extraction.

The defocus system described in this chapter also uses multiple sensors, but they share an optical axis using beam splitters. This avoids view dependence problems associated with stereo sensors (e.g., reflections, specular highlights, occlusions) and allows for an overall smaller camera. The defocus matting system pulls a trimap using defocus and uses information from all of the cameras

during matting. In this chapter, I concentrate on the matting problem instead of the trimap problem. My trimap region processing is simple compared to that of Zitnick et al. A hybrid trimap method combining this depth-from-defocus with their depth-from-stereo and region processing would likely be superior to either alone.

My automatic trimap extraction method extends the idea of depth detection from multiple defocussed images. Compared to Asada et al., the trimap algorithm's goal is to produce a less precise depth result (since it needs only three distinct values), but to compute that result everywhere, not just at edges. The input has about the same precision between the algorithms, in the multi-parameter sense. The trimap algorithm's input is higher precision in some ways. It has more data available for each image because those images are higher-resolution, color, and video. It is also lower precision because it uses fewer images.

The major challenge in extending Asada et al.'s work to trimaps is not the input but in producing good results with comparatively small PSFs. Asada et al. originally worked with small objects and large PSFs. (On small objects it is possible to produce large PSFs because they are nearly the size of the aperture.) For the matting problem it is necessary to work with human-sized figures and backgrounds containing cars and trees. For these it is hard to produce very large PSFs; one needs either a huge aperture, very high resolution, or an actor close to the camera. None of these are appropriate, and it is unacceptable to perform matting only on small objects. So reconstructing crude depths from small PSFs, which has not previously been reported, is an important part of creating trimaps for matting from defocus.

Others have recently built systems that can produce a set of differently-focussed images as output. Ng et al. [67] describe a still-camera that captures a low-resolution light field and then produces a series of higher-resolution defocussed images using Fourier methods [66]. Cohen also described a new unpublished still camera for capturing multi-focus images in a recent talk citeCohen05. To the best of my knowledge, Nayar et al. was the first (and only other group) to build multi-focus *video* camera [62], and I am the first to create a color camera with three different simultaneous focus parameters. In the next chapter I take this camera to the extreme and capture eight simultaneous focus parameters in color.

8.3 Defocus Splitting Tree

For all three algorithms: DDM, trimap extraction, and DVM; the tree in figure 8.2 is used. However, DDM does not exercise the background-focussed branch. This tree is the one implemented by the capture system pictured in figure 7.4 of the previous chapter.

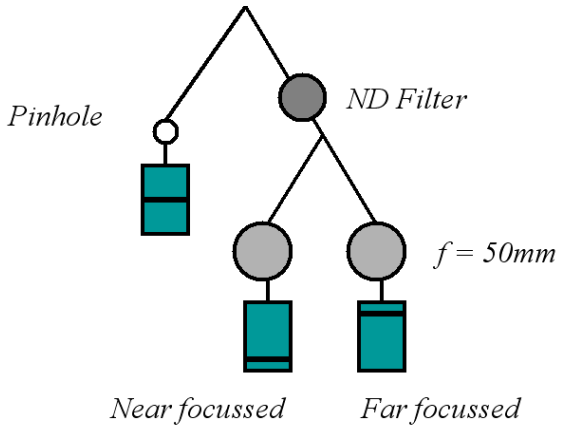


Figure 8.2: Splitting tree used for defocus matting.

Note that the distribution property of the splitting tree is employed; in the abstract tree there is a single neutral density filter, but for the actual implementation two filters are used but they are closer to the leaves.

8.4 Known Static Background

Matting is a classic problem in both computer graphics and vision. Practical solutions that yield high quality results are important for special effects in the movie industry. In the defocus difference matting (DDM) approach, I assume that the background is known, typically because it is static and pre-recorded. In this respect, our work is related to blue-screen matting. Two common problems with blue-screen matting are the limitations imposed on the color of the foreground, e.g., the actor cannot wear a blue shirt, and more importantly a color spill of the background on the foreground, which considerably changes the lighting of the scene. Our method alleviates both of these problems.

DDM is different from background subtraction because it operates on defocus (color derivative)

differences instead of image differences, which allows it to pull mattes even at pixels where the foreground and background colors are identical. When the background is known, pulling the matte is still an underconstrained problem. Smith and Blinn's triangulation method [82] used two different backgrounds, which is not practical for live subjects. I instead constrain the solution by using two video streams that share a common center of projection but different focus characteristics.

I have obtained excellent results for synthetic images with both patterned and natural backgrounds. For real images the results are acceptable but not ideal. This is because of the primary drawback of DDM; it is very sensitive to color and alignment calibration of the cameras, which I currently achieve only by physical adjustment. As future work I intend to make the method more robust by adding software correction via optical flow estimation, which has been applied successfully in other matting algorithms.

8.4.1 Algorithm

Assume a scene containing a foreground object whose image is αF and background whose image is B. The image formation equations for a pinhole camera (I_1) and a narrow-depth-of-field camera focussed on the foreground (I_2) are well-approximated by:

$$I_1 = \alpha(F - B_1) + B_1 \tag{8.2}$$

$$I_2 = \alpha(F - B_2) + B_2 \tag{8.3}$$

The foreground appears identical in these images and the background differs only by defocus. The camera has previously captured images of the background by both cameras, then B_1 and B_2 are known and it is possible to solve directly for the matte and matted foreground:

$$\alpha = 1 - \frac{I_1 - I_2}{B_1 - B_2} \tag{8.4}$$

$$\alpha F = I_1 + (\alpha - 1)B_1 \tag{8.5}$$

When the difference between the two background images is small, these equations are ill-conditioned. A post process detects pixels where the result is ill-conditioned and replaces the matte at those pixels with a value interpolated from well-conditioned neighbors.

8.4.2 Post Processing

For real data miscalibration can introduce noise into the $\alpha \approx 1$ regions, so a post-processing pass improves region coherence. The $\alpha \approx 0$ areas are stable because camera 2's background circle of confusion was chosen to be twice the highest background frequency, which itself should ideally be less than half the sampling frequency (resolution) of camera 1.

8.4.3 Results

For perfectly calibrated data, defocus difference matting can obtain excellent results. Figure 8.3 shows results on synthetic data composed from real photographs (the defocus is from the real camera). The top row shows the input images and backgrounds. To create these synthetic images, the actor was photographed in-focus with a lens camera and a large aperture. The actor and matte were pulled manually. The background object is digitally generated color noise. The mean is gray and the colors vary uniformly from black to white along each color channel. This background was projected on a white wall and photographed with a pinhole and a lens camera. Different cameras were used to image the foreground and background, so the camera parameters are not precisely identical. This allows ground truth for the experiment.

The center row of the figure shows the recovered matte and foreground with ill-conditioned pixels marked red. The bottom row of images matte and foreground reconstructed from well-conditioned samples.

8.5 Trimaps from Defocus

A *trimap* segments a pinhole image into three mutually exclusive and collectively exhaustive regions expressed as sets of pixels. These sets limit the number of unknowns and steer initial estimates. Hand-painted trimaps are common in previous work; I instead produce them automatically as follows.

Areas in the scene that have high-frequency texture produce high-frequency image content in I_P and exactly one of I_F and I_B. I use this observation to classify pixels with high-frequency neighborhoods into three regions based on the z values for which they appear sharp, as shown in the

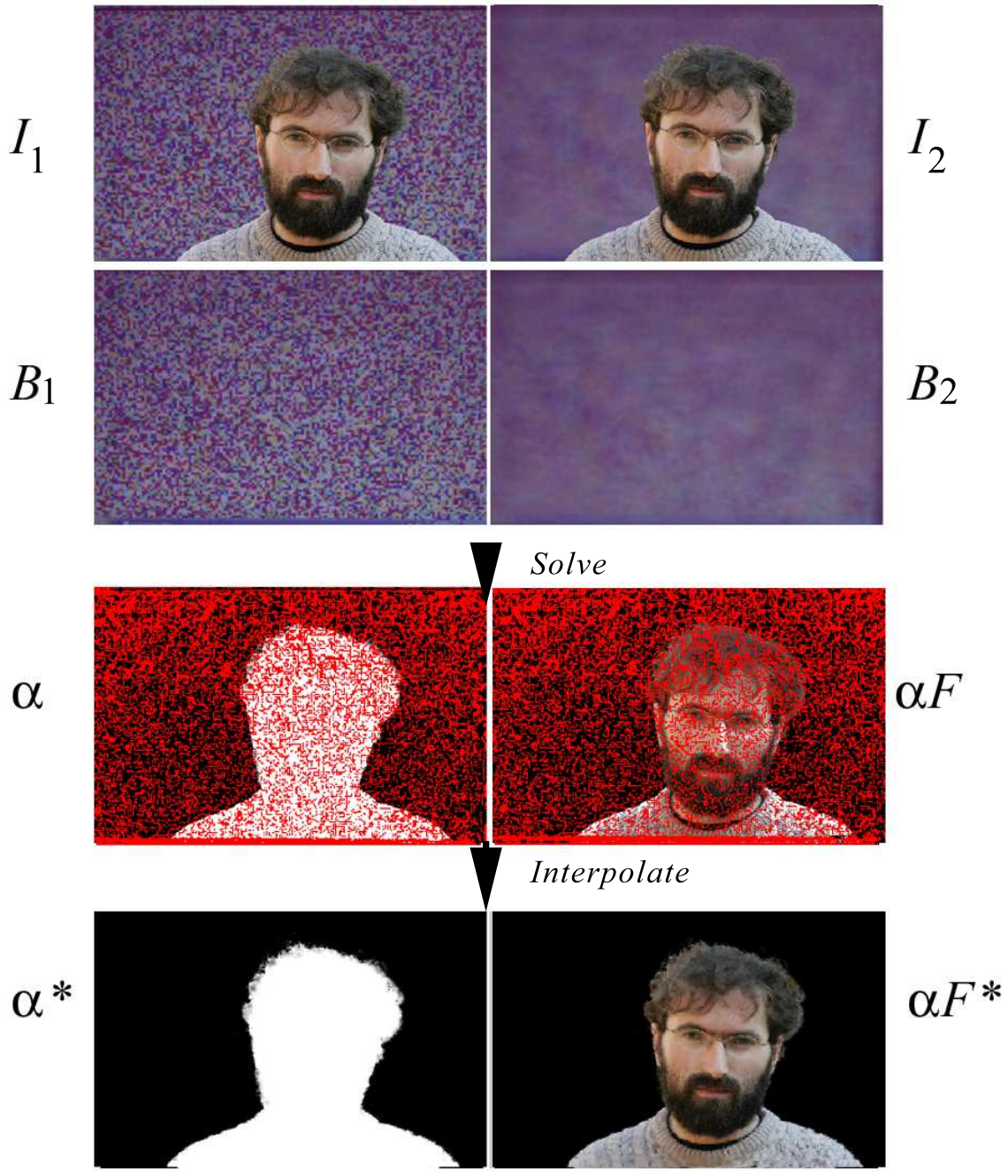


Figure 8.3: Input and results for defocus difference matting

Figure 8.4: Input and result for a synthetic image with a natural background and for a real image with a striped background.

sample trimap in Figure 8.5. Sets Ω_B and Ω_F contain pixels that are respectively "definitely background" ($\alpha = 0$) and "definitely foreground" ($\alpha = 1$). Set Ω contains "unknown" pixels that may be foreground, background, or some blend. This is the set over which I solve for the matte.

Many surfaces with uniform macro appearance actually have fine structural elements like the pores and hair on human skin, the grain of wood, and the rough surface of brick. This allows us to detect defocus for many foreground objects even in the absence of strong macro texture. It is necessary to use lower thresholds to detect high frequencies in the background, where only macro texture is visible.

The algorithm first creates classification of the foreground and background regions by measuring the relative strength of the spatial gradients:

$$\text{Let } D = \text{disk}$$

$$\text{Let } D = \text{disk}(\max(r_F, r_B))$$

$$\Omega_{F1} = \text{erode}(\text{close}((|\nabla I_F| > |\nabla I_B|) \otimes D > 0.6, D)), D) \tag{8.6}$$

$$\Omega_{B1} = \text{erode}(\text{close}((|\nabla I_F| < |\nabla I_B|) \otimes D > 0.4, D)), D)\,, \tag{8.7}$$

where erode and close are morphological operators [40] used to achieve robustness. The disk should be approximately the size of the PSFs. The algorithm then classifies the ambiguous locations either in both Ω_{F1} and Ω_{B1} or in neither:

$$\Omega = (\tilde{\Omega}_{F1} \cap \tilde{\Omega}_{B1}) \cup (\Omega_{F1} \cap \Omega_{B1}). \tag{8.8}$$

Finally, enforce the mutual exclusion property:

$$\Omega_F = \Omega_{F1} \cap \tilde{\Omega} \tag{8.9}$$

$$\Omega_B = \Omega_{B1} \cap \tilde{\Omega}\,. \tag{8.10}$$

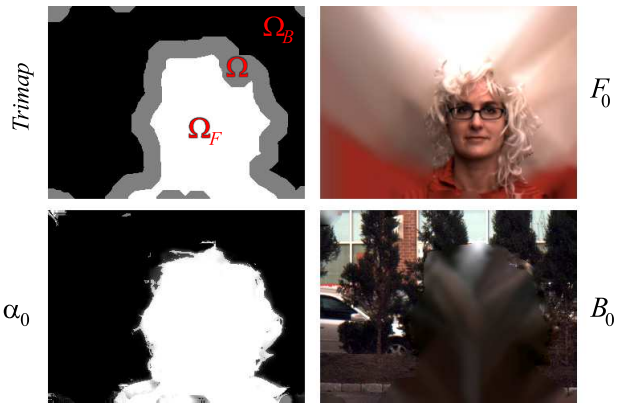


Figure 8.5: Initial estimates for defocus video matting.

Clockwise from upper left: Trimap from defocus, foreground "outpainted" to fill space, background reconstructed from adjacent frames and inpainted, and matte via pinhole composition.

8.6 Unconstrained Case

The trimaps computed in the previous section are good, but not tight enough for previous algorithms like Poisson matting to use for pulling mattes from arbitrary video. The problem is partly that the trimap generation algorithm uses wide filters that provide an over-conservative estimate of Ω. This same problem also occurs for the Poisson algorithm when a user generates the trimap manually, however, in that case the user can also recognize a poor result and adjust the trimap appropriately. The Poisson algorithm was intended to be used as part of a tool workflow, not as an autonomous system.

Recall that the ideal algorithm is not only autonomous but operates on scenes where the foreground and background are both unknown and potentially dynamic. To approach this ideal result it

is necessary to use a larger and more robust set of relations between input and output than Poisson matting's assumption that $\nabla\alpha = \nabla I$.

Defocus Video Matting (DVM) relates input and output using the full reverse-projection blurring model described in chapter 3 and operates on multi-parameter video. This richer imaging model and data set provide enough information to solve the matting problem autonomously for natural scenes. Because a blurring model of defocus is central to the algorithm, DVM requires some high frequency texture in the scene. However, it can succeed even if that texture is not present everywhere in the video. This complements Poisson and Bayesian matting, which assume a scene with primary low-frequency texture everywhere.

Like previous matting approaches, DVM poses matting as a error minimization problem and has a constrained, non-linear optimizer at its heart. However, to accommodate the multi-parameter input stream, the form of the optimizer and the error function differ significantly from previous techniques. The output is a solution to the matting problem. For each frame, the algorithm produces a model of the scene as a color foreground image F and a scalar matte image α.

In addition to recovering the traditional F and α values, DVM is also able to reconstruct the color background image B. It even produces values for the background in areas that are occluded by the foreground. The fully occluded parts of the background are reconstructed from adjacent frames and by Laplace smoothness constraints similar to those used in Poisson matting. A fascinating property of the image produced by a lens camera is that when the background is in focus, the edges of the foreground object are partially transparent due to defocus. This allows a lens camera to record information about points on the background that would be occluded in the image produced by a pinhole camera. The amount of background observed is proportional to the aperture radius. Asada et al. [**?**], Isaksen et al. [45], Favaro and Soatto [31], and others have previously noted that this property effectively allows a lens camera with a large aperture to see slightly behind a foreground object at its edges, and allows a camera with a huge aperture to see *through* the foreground everywhere. We exploit this property for the first time as part of a matting algorithm and use it to accurately recover the partly occluded background pixels. The background reconstruction is of course limited. It is lower quality than the extracted foreground image and is inaccurate in cases where background

pixels are never observed. Those cases are where the apertures used in capturing the input are fairly small and where the foreground object never moves to reveal some background locations.

The error function minimized by the optimizer is the sum-squared difference between the real multi-parameter video and one rendered from the scene model. The rendered multi-parameter video is a function of the camera parameters, the depths of the foreground and background planes, and the values of the pixels in the foreground, matte, and background images. The camera parameters are known and recorded at capture time. The approximate depths of the planes are also known at capture time (since without them, the camera cannot be properly focussed!). The free variables are therefore only the pixel values, which are constrained to the range $[0,1]$ for each channel.

In practice, the foreground and background are rarely perfect planes and need not be. For the approximations used in the algorithm to hold, the foreground and background objects need only be constrained to lie within the foreground and background camera depth fields. Because depth of field is related hyperbolically to depth, the background depth field may even stretch to infinity.

8.6.1 Optimization Setup

The defocus video matting algorithm is straightforward: minimize the error function. The derivation is interesting because the error function is based on image composition equations. Complexity arises in the implementation because the optimization problem is huge and requires some mathematical sophistication to solve efficiently. An optimization problem is characterized by the input, unknowns, error function, and constraints.

Input: As before, let $I[x,y,\lambda,...]$ be a frame of multi-parameter video. This is the input to the matting problem. It is captured using a multi-parameter camera. A discussion of exactly which camera parameters are sampled for I follows in the next subsection. For now all that is important is that there are many of them. Let each frame have K pixels and let there be M samples per pixel. Let $\vec{I}$ be the column vector of length MK that is the unraveling of I.

Unknowns: Let $\alpha[x,y]$, $F[x,y,\lambda]$, and $B[x,y,\lambda]$ be the unknown matte, foreground, and background images for a single frame of video. Each image has a total of N pixels. Assume that the

foreground and background have three color samples at each pixel; I discuss higher sampling density in chapter 10. Let u be the column vector of length $N+3N+3N=7N$ that packs together the unraveled unknowns:

$$u = \begin{bmatrix} \vec{\alpha} \\ \vec{B} \\ \vec{F} \end{bmatrix} \tag{8.11}$$

i.e., u is a compact representation of the scene. The input and outputs have different sizes because the matting algorithm uses defocus. Defocus effects have wide filters, so some pixels at the edge of the image frame will not appear in the output because the filter taps fall outside the input. Thus $N < M$.

Error Function: Let image composition function $C(\cdot)$ generate a multi-parameter image $C(\alpha,B,F)[x,y,\lambda,...]$ of the scene. The parameters of $C(\cdot)$ must exactly match those of I, so that when α, B, and F are an accurate model of the real scene, $C(\alpha,B,F)=I$. Any photorealistic renderer, such as a ray tracer, can be used to implement $C(\cdot)$. In practice DVM uses a highly optimized renderer that exploits the simplicity of the scene.

Let $\vec{C}(\cdot)$ be the corresponding unraveled function that instead maps an unraveled scene to an unraveled multi-parameter image. Like $\vec{I}$, column vector $\vec{C}(u)$ has length MK.

The goal of matting is to find the input scene that reproduces the input image under the composition equations. Let error function $\vec{E}(u)$ produce the length-MK vector of differences between the unraveled composite and input,

$$\vec{E}(u) = \vec{C}(u) - \vec{I}. \tag{8.12}$$

Let scalar-valued function $Q(u)$ be the norm of the error vector,

$$Q(u) = \sum_k \frac{1}{2}\vec{E}_k^2(u). \tag{8.13}$$

Later in this section, Q is extended with small-magnitude regularization terms that help stabilize the algorithm in the presence of ambiguous input.

To simplify the notation, functions Q, $\vec{E}$, and $\vec{C}$ are frequently written without the u argument as if they were vectors.

Because $\vec{I}$ is known at the beginning of the problem, evaluating $\vec{E}(u)$ and $Q(u)$ is only as computationally hard as evaluating $\vec{C}(u)$. This chapter therefore discusses the cost of $\vec{C}(u)$ exclusively and assumes that the subtraction and summation for the error metrics are negligible compared to the cost of rendering an image from the scene data.

In optimization terminology, there are two kinds of constraints. The error vector is a set of constraints, in the sense that it gives rise to a set of equations that limit the possible values for the unknowns. There are also so-called hard constraints about the individual values.

Constraints: All elements of α are inherently constrained to the unit interval $[0,1]$. All elements of images F, B, and I must be non-negative, since images are a measure of radiance. If I is a properly exposed and normalized image, then all of its elements are less than 1. In this case, it is likely that all elements of F and B are also less than 1. However it is possible for them to be outside the range yet produce a properly exposed image since F is always modulated by α. Thus $u \geq 0$ is a hard constraint in all cases, but $u \leq 1$ is a hard constraint only if the scene is known to be well within the dynamic range of the camera.

The solution to the matting problem is a scene vector $u^\star$ that minimizes $Q(u)$:

$$u^\star = \operatorname*{argmin}_u Q(u)\,. \tag{8.14}$$

This equation has $7N$ unknowns and MK constraints. The problem is constrained at $7N = MK$, overconstrained when $7N < MK$ and underconstrained when $7N > MK$. For the matting problem, it is best to overconstrain the problem since local areas of image data are often ambiguous with respect to focus and the data are likely noisy as well. Overconstraining is accomplished by increasing the precision of the multi-parameter image I. Assuming conventional video sensors with three wavelength samples per pixel (RGB) and focus filters that are not too wide ($K \approx N$), a minimum of three sensors ($M = 3 \times 3 = 9$) are required to overconstrain the system of equations.

Regardless of the choices of camera parameters and method for computing C, the image composition equations in C must contain terms of the form αF. Thus the norm of the error must (at least) contain terms of the form $(\alpha F)^2$. In unraveled notation, these terms are $(u_i u_{i+3jN})^2$. Index $4 \leq j \leq 6$ arises because the unraveling packs $\vec{\alpha}$ first and follows it by the color channels of B and F. Ignoring

the subscript relations, Q must generally contain fourth-order terms in u. Function Q is therefore non-linear in the unknowns, and is in fact quartic. The optimization problem at hand is therefore non-linear and constrained with regard to the unknowns. It is also important and intentional that the form of the error function is a summation of squares (of non-linear functions) because that form has been studied extensively in the optimization literature.

The minimization problem is also large. For 640×480 color video, $N = 640 \times 480 \times 3 = 921,600$. Equation 8.14 therefore holds about 6 million unknowns and 8 million constraints. The previous section demonstrated that trimaps can be obtained automatically using defocus information from a multi-parameter image. A typical trimap reduces the number of unknown pixels by about 80%, so that only about 150,000 unknowns need to be considered per frame. To further reduce the size, it is possible to solve the matting problem on small subregions (typically, blocks) of an image and then resolve inconsistencies at the borders of the blocks. However, using subregions makes the problem harder to solve because less information is available at each block and the ratio of border pixels to interior pixels increases with the number of regions. In practice, the problem still presents at least 15 thousand unknowns and closer to 20 thousand constraints, which is considered a large non-linear problem.

It may help the reader to be aware of the following notation conventions: C is a mnemonic for "constraints" or "composite," u is for "unknowns," and I is for "input" or "image." Index $0 \leq n < 9N$ appears exclusively as a subscript for unknowns (u) and index $0 \leq k < 9K$ is for constraints ($\vec{E}, \vec{C}, \vec{I}$).

8.6.2 Solver

There are many[1] iterative methods for solving equation 8.14. All non-linear solvers begin with an initial estimate for u and perturb that guess in successive iterations until some convergence criteria is met. In many cases it is a good idea to begin with several initial estimates to avoid local minima. The solvers differ in the amount of memory (and time) that they consume per-iteration, their convergence properties for certain kinds of error functions, and whether they require explicit derivatives of the error function.

[1] See Numerical Recipes [75] chapter 10 for a detailed discussion of non-linear solvers.

Most solvers require either linear or quadratic memory, where the quadratic-space methods are generally forming locally linear systems and storing them as matrices. Even when the problem has been reduced by a trimap and subregions, with tens of thousands of unknowns and constraints stored in 32-bit floating point format, a solver using quadratic storage requires more than a terabyte of memory! A solver requiring quadratic storage in the size of the problem therefore appears unacceptable. Yet there are two subtleties of the quadratic storage cost. First, the matrix may be sparse, so the quadratic-space requirement may only hold for solver ignorant of the structure of the constraints. Second, if the solver inverts the matrix, the inverse of an arbitrary sparse matrix is dense and the storage problem rises again. In this latter case, the $O(N^3)$ time cost of computing the dense inverse is likely even more unacceptable than the storage cost of the result. So there are really three cases of significance regarding storage: linear-space (good), sparse quadratic-space (acceptable), and dense quadratic-space (unacceptable).

Conjugate gradient descent methods, particularly the popular Gauss-Newton and Levenberg-Marquardt [37, 49, 52] variations, are in the category that requires quadratic space to store the dense pseudo-inverse of a matrix when implemented naively. I experimented with the Matlab implementation of the Levenberg-Marquardt optimizer for defocus video matting and found that the convergence properties were indeed attractive. However, only uselessly small regions would fit in memory, so I abandoned that line of research in favor of the lighter-weight gradient descent method. It is possible to implement these more efficiently, as is discussed in the future work chapter.

Gradient descent is generally considered a poor solver because of its convergence properties. However, it has ideal space and time behavior. Where derivatives of the error function are available and can be expressed as a sparse matrix, gradient descent requires only linear space and time per iteration.

A gradient descent solver begins at a current scene u and chooses a new scene $u+\Delta u$, where Δu

is opposite the gradient of $Q(u)$ in order to descend quickly towards the minimum:

$$\Delta u = -\nabla Q \tag{8.15}$$

substituting for the gradient,

$$= -\nabla \sum_k \frac{1}{2}\vec{E}_k^2 \tag{8.16}$$

so that a given element is

$$\Delta u_n = -\frac{\partial \sum_k \frac{1}{2}\vec{E}_k^2}{\partial u_n} \tag{8.17}$$

$$= -\sum_k \left(\vec{E}_k \frac{\partial \vec{E}_k}{\partial u_n} \right). \tag{8.18}$$

Equation 8.18 contains the vector dot product $\vec{E} \cdot \frac{\partial \vec{E}}{\partial u_n}$. It can be written for all n as a vector-matrix product using the matrix of all $\vec{E}$ partial derivatives. Such a matrix is known as the *Jacobian*. Let $\mathbf{J}(u)$ be the function that computes the Jacobian matrix for $\vec{E}(u)$. It contains the partial derivative of each element of $\vec{E}(u)$ with respect to each element of u such that

$$\mathbf{J}_{k,n}(u) = \frac{\partial \vec{E}_k(u)}{\partial u_n}. \tag{8.19}$$

For the matting problem, the size of the Jacobian is $9K \times 7N$. Using $\mathbf{J}$ gives a compact expression for the direction of steepest descent,

$$\Delta u = -\vec{E}^T \mathbf{J}. \tag{8.20}$$

When explicit derivatives are not available, it is common practice to compute $\mathbf{J}$ by numerical differentiation:

$$\mathbf{J}_{k,n}(u) = \frac{\vec{E}_k(u+\delta) - \vec{E}_k(u-\delta)}{\varepsilon}. \tag{8.21}$$

where δ a small step in the nth dimension:

$$\delta_i = \begin{cases} \varepsilon/2 & i = n \\ 0 & \text{elsewhere} \end{cases} \tag{8.22}$$

With the Jacobian thus defined, a basic gradient descent algorithm for matting is as shown in Figure 8.6.

1. Choose a random u vector
2. While $Q(u) >$ threshold
   (a) Let $\Delta u = -\vec{E}^T(u)\mathbf{J}(u)$
   (b) $u \leftarrow \max(\min(1, u + (\text{step})\Delta u, 0)$
3. Let $u^\star = u$
4. Unravel and unpack α, B, and F from $u^\star$

Figure 8.6: Matting algorithm with a gradient descent solver.

Step and *threshold* are scalar constants that control the convergence properties; they are discussed later in this chapter. The *min* and *max* functions enforce the hard constraints on u at each iteration.

The structure of the algorithm is fine as given (although later in this chapter I use generated trimaps to reduce the size of the problem and choose the initial u, and post-process $u^\star$ to remove artifacts). The problem with implementing the algorithm as stated is that computing $\mathbf{J}$ by numerical differentiation in step 2a is too expensive. It would require $7 \times 9 \times 2NK = 126NK = O(N^2)$ evaluations of $\vec{E}(u)$ as presented. Computing the error function requires non-zero time (it must compute a photorealistic image $\vec{C}(u)$!), and even if it were trivial to compute, the quadratic space and number of error function calls required is prohibitive. Thus computing $\mathbf{J}$ numerically undoes the claimed space and time advantage of gradient descent.

Fortunately, the quadratic space and time bounds for computing Δu are upper bounds. When $\mathbf{J}$ is sparse with known structure and $\vec{C}(\cdot)$ is symbolically differentiable, both bounds are linear in the number of non-zero elements of $\mathbf{J}$. The as a whole, composition function must have at least $7N$ terms because it uses[2] every element of u. Thus the computation time and space for $\mathbf{J}$ is bounded below by $\Omega(N)$ and above by $O(N^2)$. The linear lower bound is of course preferable. This guides the choice of composition function; it is desirable that $\vec{C}(\cdot)$ have the following properties:

- Three samples per pixel (to produce $9K$ constraints),

[2] Technically, the composition function may ignore some elements of the background that are occluded and elements of the foreground where $\alpha = 0$, but at the start of the problem α is itself unknown so the composition function is general and no input elements can be ignored.

- Exactly $\Theta(N)$ non-zero terms (... which implies a constant number of terms per pixel),
- Efficient to compute (small constants in the asymptotic bound), and
- Easily differentiable (to compute **J**).

I previously deferred defining how $\vec{C}$ is implemented, offering only bi-directional ray tracing as a straw-man. It should be clear at this point that ray tracing is in fact a poor way of computing $\vec{C}(u)$, since it has none of the desirable properties!

For the matting problem, a much more efficient method is available using Fourier model of image formation described in chapter 3. The next section describes a choice of camera parameters and image model that leads to a good $\vec{C}(\cdot)$. It is no accident that the chosen camera parameters effectively vary the amount of defocus– the defocus difference algorithm for solving matting in the known background case and the automatic trimap generator have already shown that the defocus composites are compact and useful. Also, in order to use the trimap generator to reduce the size of the general matting problem for defocus video matting it is necessary to use compatible multi-parameter videos as input for both algorithms.

8.6.3 Defocus Composites

Let $a[x,y]$ be a 2D matrix, $F[x,y,\lambda]$ and $B[x,y,\lambda]$ be 3D matrices. We generalize the two-plane compositing expression with a function of the scene that varies over two discrete spatial parameters, a discrete wavelength (color channel) parameter λ, and a discrete focus parameter $z \in \{1,2,3\}$:

$$C(\alpha,F,B)[x,y,\lambda,z] = (\alpha F[\lambda]) \otimes h[z] + (1 - \alpha \otimes h[z])(B[\lambda] \otimes g[z])\big|_{[x,y]}, \tag{8.23}$$

where 3D matrices h and g encode the PSFs:

$$h[x,y,z] = \begin{cases} \delta[x,y], & \text{z=1} \\ \text{disk}(r_F)[x,y], & \text{z=2} \\ \delta[x,y], & \text{z=3} \end{cases} \tag{8.24}$$

$$g[x,y,z] = \begin{cases} \delta[x,y], & \text{z=1} \\ \delta[x,y], & \text{z=2} \\ \text{disk}(r_B)[x,y], & \text{z=3}\,. \end{cases} \tag{8.25}$$

Constants r_F and r_B are the PSF radii for the foreground and background planes when the camera

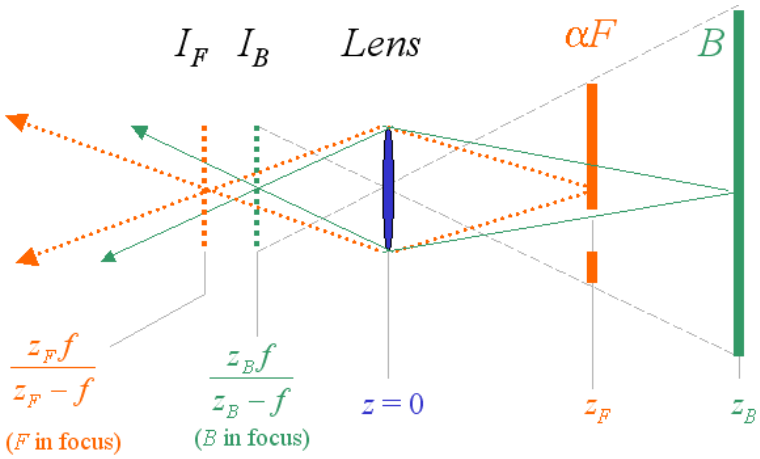


Figure 8.7: The point spread function is the intersection of a cone and the imager.

is focused on the *opposite* plane.

Equation 8.23 can be evaluated efficiently: for small PSF radii, we can simulate a 320×240 lens camera image in ten milliseconds.

8.6.4 Exact Differentiation of $\vec{E}$

I now derive a simple expression for the elements of the Jacobian from the chosen defocus $\vec{C}(\cdot)$. This expression shows that the Jacobian is sparse, so that computing Δu in equation 8.20 is feasible.

By definition, the elements of the Jacobian are:

$$\mathbf{J}_{k,n} = \frac{\partial(\vec{C}_k(u) - \vec{I}_k)}{\partial u_n} = \frac{\partial \vec{C}_k(u)}{\partial u_n}. \tag{8.26}$$

To evaluate this, expand the convolution from equation 8.23. It is necessary to change variables from packed 1D vectors indexed by k back to multi-parameter images indexed by x (and y, but I only show a single spatial parameter to improve readability):

$$C[x,z,\lambda] \;=\; \sum_s \alpha[s]F[s,\lambda]h[x\;-\;s,z] \;+\; \left(1 - \sum_s \alpha[s]h[x-s,z]\right) \sum_s B[s,\lambda]g[x\;-\;s,z]\;. \tag{8.27}$$

Examination of this expansion shows that $\mathbf{J}$ is both sparse and simple. For example, consider the case where unknown u_n corresponds to $F[i,\lambda]$. In a full expansion of equation 8.27, only one term

contains $F[i,\lambda]$, so the partial derivative contains only one term:

$$\frac{\partial C[x,\lambda,z]}{\partial F[i,\lambda]} = \alpha[i]h[x-i,z]\,. \tag{8.28}$$

The expressions for α and B derivatives are only slightly more complicated, with (potentially) non-zero elements only at:

$$\begin{aligned}\frac{\partial C[x,\lambda,z]}{\partial \alpha[i]} &= h[x-i,z]\left(F[i,\lambda]-\sum_s B[\lambda,s]g[x-s,z]\right)\\ &= h[x-i,z]\left(F[i,\lambda]-(B[\lambda]\otimes g[z])[x]\right)\end{aligned} \tag{8.29}$$

$$\begin{aligned}\frac{\partial C[x,\lambda,z]}{\partial B[i,\lambda]} &= g[x-i,z]\left(1-\sum_s \alpha[s]h[x-s,z]\right)\\ &= g[x-i,z]\left(1-(\alpha\otimes h[z])[x]\right)\,.\end{aligned} \tag{8.30}$$

Note that the summations in these last two cases are just elements of convolution terms that appear in $\vec{E}$, so there is no additional cost for computing them. Figure 8.8 shows the structure of an actual Jacobian matrix computed in Matlab and rendered with the *spy* function. Background colors in the diagram represent the color channel corresponding to each term. Dark blue marks non-zero elements. The non-zero elements are in blocks along diagonals because of the unraveling process. Each column is effectively an unraveled image of several PSFs.

8.6.5 Trust Region and Weights

The gradient gives the direction of the step to apply to u to reduce the error function. I use a so-called dogleg trust region scheme (see [68] for discussion) to choose the magnitude of the step.

The idea is to take the largest step that also most decreases error. Begin with a trust region of radius $S=1$. Let $u' = \max(0,\min(1,u+\frac{S\Delta u}{|\Delta u|}))$. If $|\vec{E}(u')| < |\vec{E}(u)|$, then assume that the solver has not overshot the minimum and repeatedly double S until the error increases above the lowest level seen this iteration. If $|\vec{E}(u')| > |\vec{E}(u)|$, then assume that the solver has overshot and take the opposite action, repeatedly halving S until the solver passes the lowest error seen this iteration. When S becomes very small (e.g., 10^{-10}) or the error norm shrinks by less than 0.1%, the solver is may assume that it is at the local minimum and terminate the optimization process.

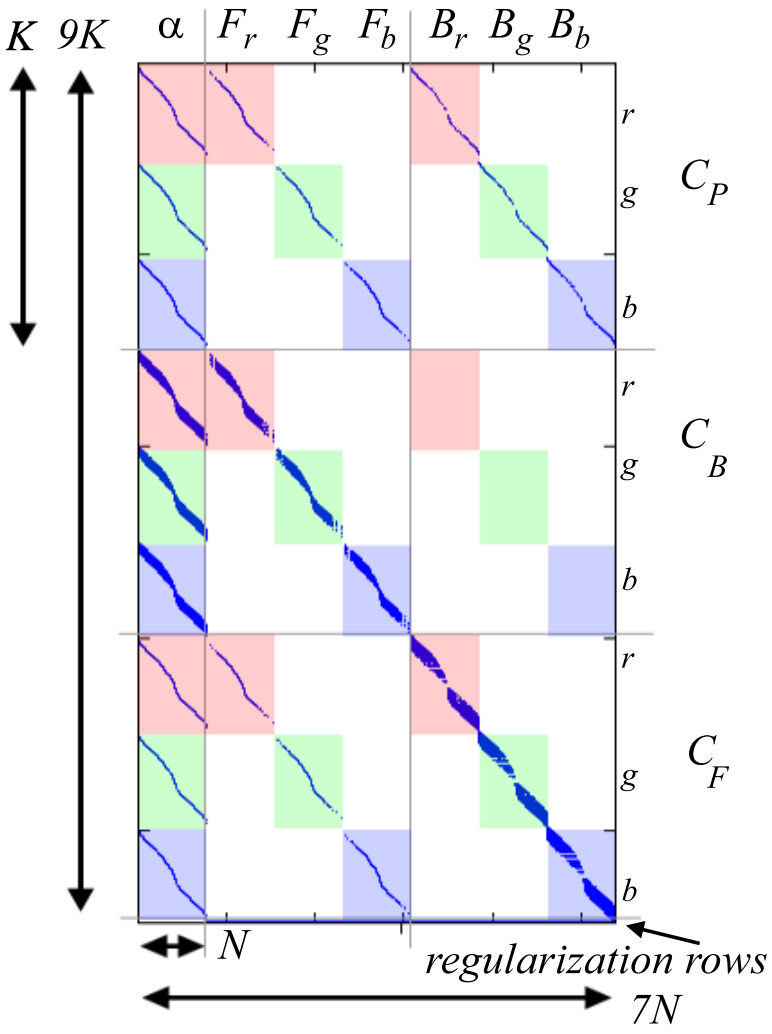


Figure 8.8: Sparsity structure of **J**.

Because the initial estimates are frequently good, the algorithm weigh the first N elements of Δu by constant $\beta_\alpha \approx 3$ to influence the optimizer to take larger steps in α. This speeds convergence without shifting the global minimum. The algorithm also reduces the magnitude of $\vec{E}$ elements corresponding to I_P by a factor of $\beta_P \approx \frac{1}{4}$. The narrow aperture and long exposure of the pinhole image produce more noise and motion blur than I_F and I_B, and this prevents over-fitting the noise. It also reduces the over-representation in $\vec{E}$ of in-focus pixels that occurs because F and B are in focus in two of the constraint images and defocused in one each.

8.6.6 Regularization

In foreground areas that are low frequency or visually similar to the background, there are many values of u that will satisfy the constraints. We bias the optimizer towards likely solutions. This is *regularization* of the optimization problem, which corresponds to having a different prior for a maximum likelihood problem. Regularization also helps pull the optimizer out of local minima in the error function and stabilizes the optimizer in areas where the global minimum is in a flat region of many possible solutions.

Extend the error vector $\vec{E}$ with p new entries, each corresponding to the magnitude of a $7N$-component *regularization vector*. Calling these regularization vectors $\varepsilon, \phi, \gamma, \ldots$, the error function Q now has the form:

$$Q(u) = \sum_k \vec{E}_k^2 = \left[\sum_{k=1}^{9K} \vec{E}_k^2\right] + \vec{E}_{9K+1}^2 + \vec{E}_{9K+2}^2 + \ldots \tag{8.31}$$

$$= \sum_{k=1}^{9K} \vec{E}_k^2 + \beta_1 \frac{9K}{7N} \sum_n^{7N} \varepsilon_n^2 + \beta_2 \frac{9K}{7N} \sum_n^{7N} \phi_n^2 + \ldots . \tag{8.32}$$

Let e be one of the regularization vectors. Each summation over n appears as a new row in $\vec{E}$ and $\mathbf{J}$ for some $k > 9K$:

$$\vec{E}_k = \left(\beta \frac{9K}{7N} \sum_n e_n^2\right)^{\frac{1}{2}} \tag{8.33}$$

$$\mathbf{J}_{k,n} = \frac{\partial \vec{E}_k}{\partial u_n} = \frac{\beta}{\vec{E}_k} \frac{9K}{7N} \sum_i \left[e_i \frac{\partial e_i}{\partial u_n}\right] . \tag{8.34}$$

The factor of $\frac{9K}{7N}$ makes the regularization magnitude invariant to the ratio of constraints to unknowns and the scaling factor β allows us to control its significance. We use small weights on

the order of $\beta = 0.05$ for each term to avoid shifting the global minimum. The outer square root in the $\vec{E}$ expression is canceled by the square in the global error function Q. In the computation of $\vec{E}^T\mathbf{J}$, the $\vec{E}_k$ factor in the denominator of Equation 8.34 cancels; in what follows, we will give, for each regularization vector, both e_n and $(\vec{E}^T\mathbf{J})_n$. We choose regularization vectors that are both easy to differentiate and efficient to evaluate: the summations over i generally contain only one non-zero term.

Straightforward but tedious differentiations lead to the expressions for $(\vec{E}^T\mathbf{J})_n$ in each of the following regularization terms; the details are omitted.

Coherence: spatial gradients are small,

$$e_n = \frac{\partial u_n}{\partial x};\ (\vec{E}^T\mathbf{J})_n = -\frac{\partial^2 u_n}{\partial x^2}\,. \tag{8.35}$$

I apply separate coherence terms to α, F, and B, for each color channel and for directions x and y. The α gradient constraints are relaxed at edges (large values of $|\nabla I_P|$) in the original image. The F gradient constraints are increased by a factor of ten where $|\nabla\alpha|$ is large. These allow sharp foreground edges and prevent noise in F where it is ill-defined.

Discrimination: α is distributed mostly at 0 and 1,

$$e_n = u_n - u_n^2; (\vec{E}^T\mathbf{J})_n = (u_n - u_n^2)(1-2u_n) \;\Big|\; 1 \le n \le N\,. \tag{8.36}$$

Background Frequencies should appear in B:

$$\text{Let } G = I_B - I_F \otimes \text{disk}(r_F)$$

$$e_n = \frac{\partial u_n}{\partial x} - \frac{\partial \vec{G}_n}{\partial x}; (\vec{E}^T\mathbf{J})_n = -\frac{\partial^2 u_n}{\partial x^2} \;\Big|\; 4N+1 \le n \le 7N. \tag{8.37}$$

8.6.7 Results on Synthetic Data

A Matlab implementation of this method pulls mattes in about seven minutes per frame, comparable to the time for a recent user-assisted matting technique [84].

Because defocus matting is unassisted and each frame is computed independently, so frames can be processed in parallel. I built a "matting farm" of eight PCs for an amortized processing time under one minute per frame. The method could someday execute in real-time; over 95% of

the solution is obtained within thirty seconds of optimization per frame and the hardware assisted trimap computation already produces a real-time preview.

I tested the method on both still images and video, and on both synthetic (rendered) and real scenes. A review of the results begin with the synthetic scenes, which allows comparison to previous work, measure deviation from a known ground truth, and to measure the effect of error sources on the result. Synthetic scenes are constructed from three images, α, F, and B using the filter-based synthesis approximation and reasonable foreground and background depths (e.g., 2m and 6m).

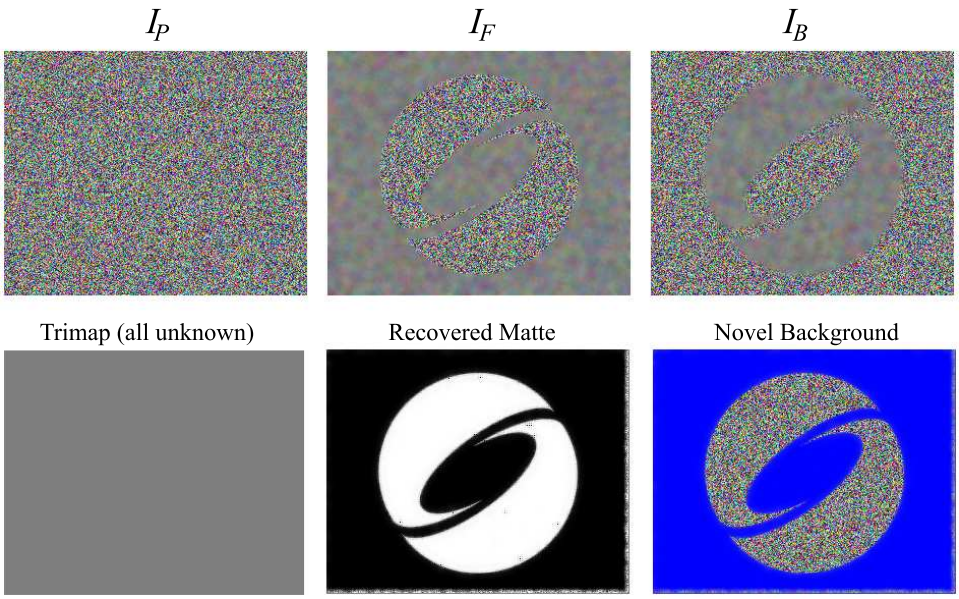


Figure 8.9: Correctly recovering the scene with no trimap, where every pixel is unknown.

Figure 8.10: Good mattes pulled in the absence of high-frequency texture.

Figure 8.9 shows a hard "corner case" in which foreground and background are both noise. This is a synthetic scene. The foreground object is the ACM SIGGRAPH created by cutting it out of a

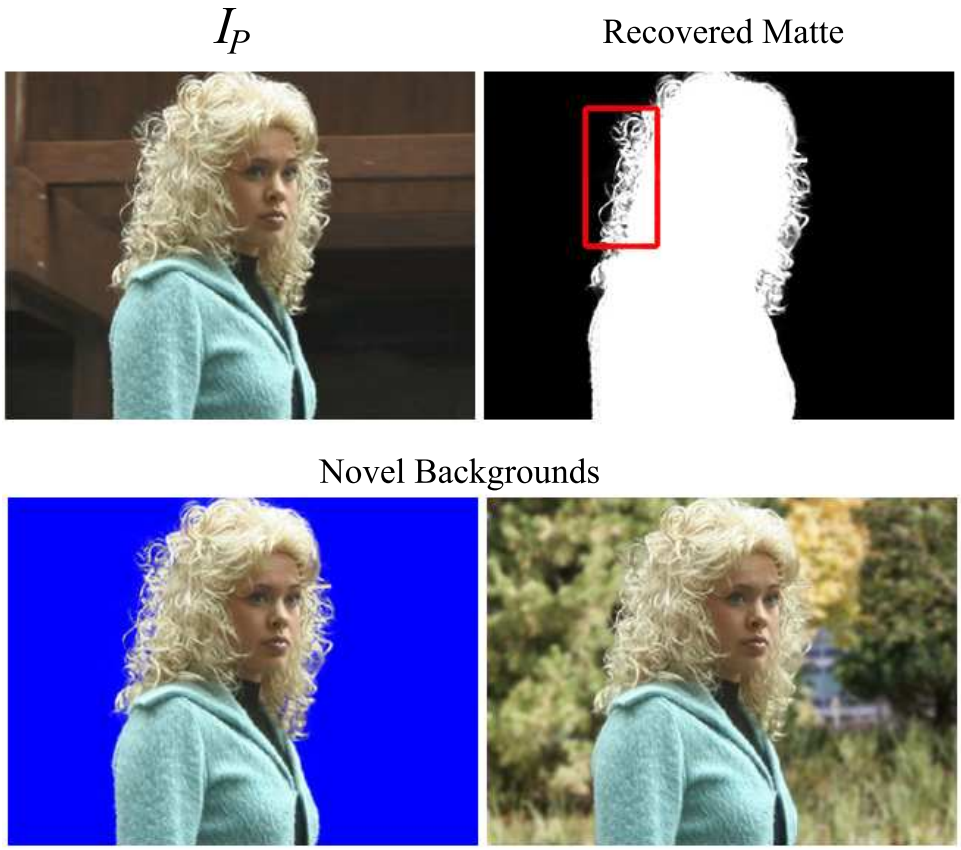


Figure 8.11: Pinhole image, recovered matte, and two recomposited images for a challenging case with fine hair structure.

different noise image. The top row shows the images seen by the three cameras. The bottom row shows a trimap that treats all pixels as unknown and the final matting result. Although the system can pull a very good trimap for this scene, I intentionally replaced the trimap with one in which the unknown region encompasses the whole image. Despite the lack of a good trimap, defocus matting is still able to pull a good matte. This demonstrates that the quality of the result is independent of the trimap, unlike previous approaches that learn a color model from the trimap. Because there is no trimap, convergence is slow and processing time was 3 hours for 320×240 pixels. Because both foreground and background have identical color distributions, this scene is an impossible case for color model approaches. Moreover, in the pinhole image, the foreground and background are indistinguishable! A matting approach working from a that image alone could not succeed because there is no information.

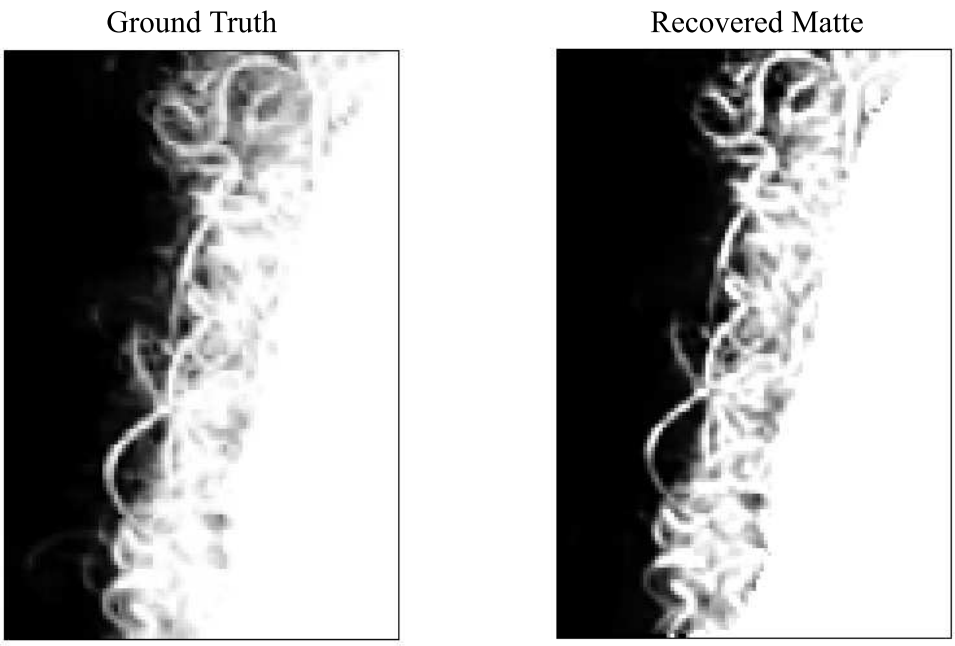


Figure 8.12: Comparison of the synthetic ground truth to the recovered scene for the hair from the inset box of figure 8.11.

Figure 8.10 shows a yellow square in front of a green screen. Neither foreground nor background contains any high-frequency texture. This is an ideal case for blue screen, Bayesian, and Poisson matting, which all assume a low frequency background. Given a hand-painted trimap, the defocus matting algorithm recovers a perfect matte as well. It cannot, however, pull an automatic trimap for this scene because of the artificial lack of texture. This demonstrates that the minimization still converges even in the absence of texture, relying mostly on regularization terms.

To compare against scenes used in previous work I use the published recovered α and αF. I reconstruct the occluded part of the background in a paint program to form B. An ideal result in this case is a reconstruction of the matte from the previous work; we cannot exceed their quality because we use their result as ground truth.

Figure 8.11 uses data that is now standard for matting papers, a difficult case with complex hair structure (data set from [20], ground truth from [84]). We pull a good matte in 30 seconds at 320×240 and 5 minutes at 640×480. The nonlinear performance falloff occurs because the smaller resolution can be solved as two large blocks while the larger requires solving many more

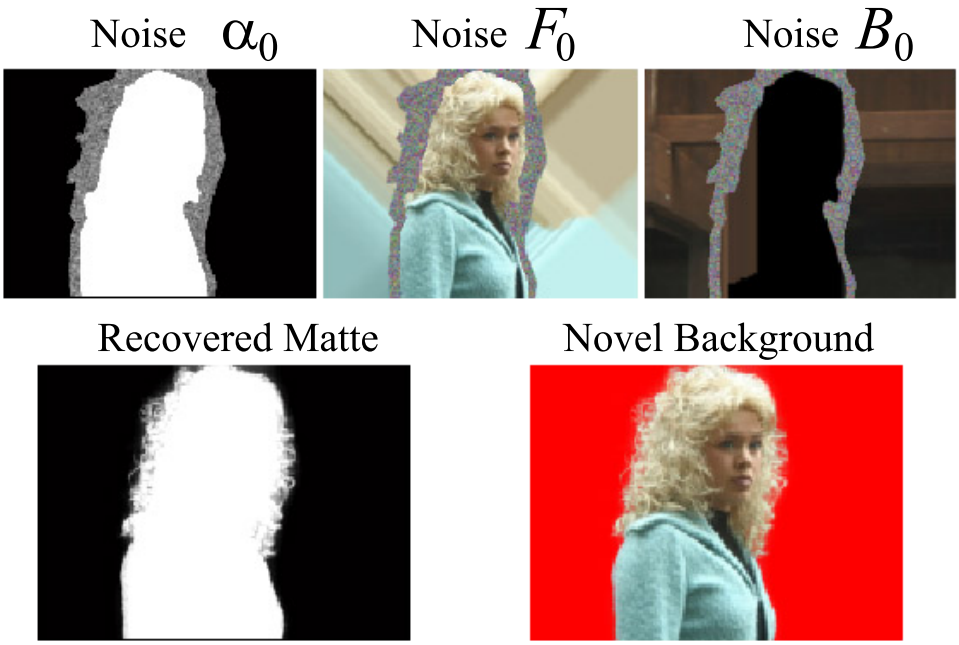


Figure 8.13: Good mattes pulled from bad (noise) initial estimates.

subproblems and letting information propagate. Figure 8.12 shows an enlarged area of the hair detail from figure 8.11. Our matte reflects the single-pixel hair structures from the ground truth structure. The contrast level is slightly higher in our matte than the ground truth matte because the regularization terms reject the broad $\alpha \approx 0.1$ areas as statistically unlikely.

Figure reffig:blond-noise demonstrates that even in the absence of good initial values it is possible to pull a good matte. Here, the foreground and background in the Ω region were seeded with noise but the algorithm still converged (albeit slowly) to a correct result. This matte took three minutes to pull. This shows that results can be independent of the quality of the initial estimates.

8.6.8 Sources of Error

There are many sources of error in a real multi-parameter images that lead to errors in the recovered mattes. The most significant are translation (parallax) error between sensors, over- and under-exposure, and different exposure times (i.e., amounts of motion blur) between cameras. These lead to pixels in different sensors that are not close to the values predicted by the composition equation. Other sources of error have less impact. Our approach appears to be robust to color calibration error,

which does not affect the gross intensity magnitudes and yields the same least-squares solution. Radial distortion by the lenses is small compared to the translation errors observed in practice due to miscalibration. DDM's gradient constraints and the inherently wide filter support of defocus tend to minimize the impact of sensor noise. Figure 8.14 shows a case where the inputs have been shifted a few pixels, one has been rotated about 2 degrees, and the colors of two have been shifted; remarkably, the algorithm still converges, although the result is not very satisfactory.

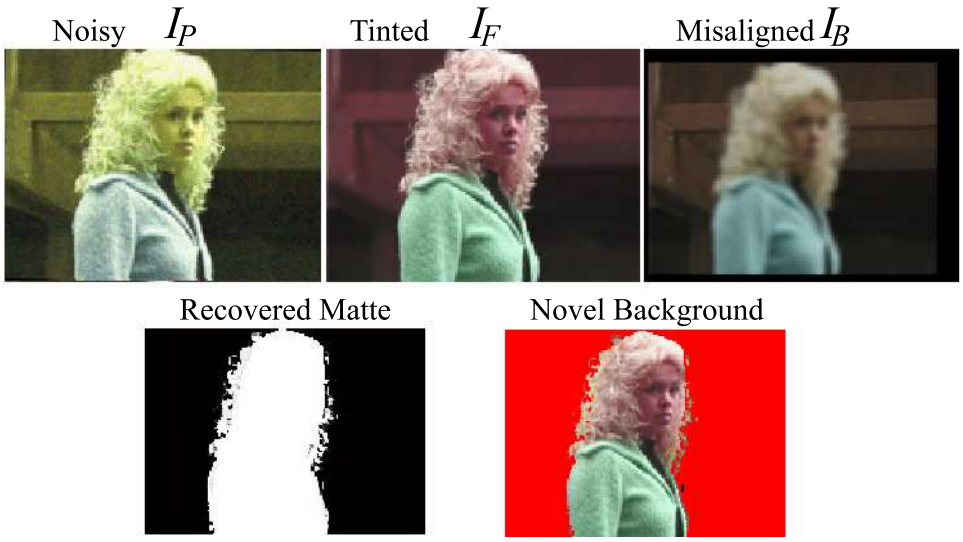


Figure 8.14: Defocus matting is robust against rotation, translation, and hue-shift.

8.6.9 Results on Real Data

In real images, the sources of error discussed in the previous section produce low-order noise near the boundaries of the trimap, as holes in the matte, and as opaque regions that should have been transparent in the matte. This can be seen in figure 8.16, where a light-colored building panel blends into over-exposed hair in the foreground during one frame of a video sequence. The "bloom" from over-exposed hair on the right is conflated with defocus; the bright building panel in the background has no texture, and happens to align with the specular highlights from the hair. Coherence terms in the regularization mislead the optimizer into classifying the panel as foreground and create an artifact too large to correct. On the left side there is also small amount of noise, at the border of Ω_B. This is correctable; we overcome small amounts of noise by modulating the recovered matte

towards black within 3 pixels of Ω_B and removing disconnected components, i.e., tiny, bright α regions that do not touch the region enclosing Ω_F. We likewise fix holes well within objects by filling the region 12 pixels inside the $\alpha < 0.1$ isocurve. The matte results in this paper are the unprocessed optimization result unless otherwise noted; in the video they are all post-processed. When the sources of error are small so that the problem is well-posed, defocus matting succeeds at pulling good mattes for real images and video.

Figure 8.17 again shows an example of matting with hair, this time from a real image. The top of the figure shows the pinhole image, extracted matte, extracted foreground, and composition over novel backgrounds. Note motion blur near the nose has been detected in the α image. The bottom of the figure shows three frames of the matte from the video of the moving dark hair. For this data set, the algorithm pulls a good matte, reconstructing both the fine hair structure and the motion blur on the left side of the face.

To experiment with automatic trimap extraction in the absence of background texture, this scene was intentionally constructed with no high frequencies in the background. I drew a trimap for the first frame and then propagated the trimap forward, so that areas well within the $\alpha = 0$ and $\alpha = 1$ regions (at least 10 pixels in) of the final matte for frame i became Ω_B and Ω_F for frame $i+1$. This works well on the hair, but cannot account for objects moving in from off-camera. It is also prone to propagate any matte errors forward. The lower part of the figure shows a selections from a good result. The images are three frames of the pulled matte from at different points in the video sequence.

We observed “checkerboard” artifacts in the matte under two circumstances. We originally used

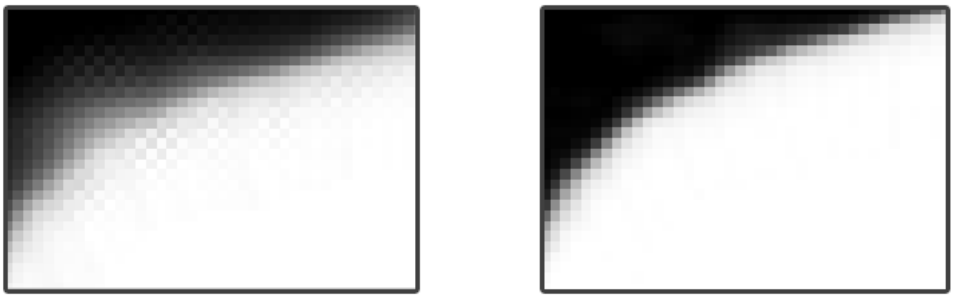


Figure 8.15: Ringing in the matte when the coherence terms are too large.

a fast Bayer interpolation that produced a half-pixel shift in the red and blue channels. Because the beam-splitters mirror the image left-to-right, this shift was inverted between cameras and led to inconsistent alpha values in a checkered pattern. Better Bayer interpolation removes this artifact. A similar but independent artifact occurs at sharp α edges when the magnitude of the α coherence regularization term is too large. Figure 8.15 shows a closeup of the matte pulled for the actor's shoulder from Figure **??**. The subimage on the left has a large magnitude ($\beta = 0.3$) α coherence term, which leads to both excessive blurring and ringing. The ringing is a checkerboard because horizontal and vertical derivatives are handled separately. The subimage on the right shows the correct matte pulled with a reasonable ($\beta = 0.05$) coherence magnitude.

Figure 8.18 shows three frames of video on the left and their automatically extracted trimaps on the center. Pinhole images are shown on the left, extracted trimaps at the center, and the final, post-processed mattes pulled are on the right. The foreground and background-focussed images are not shown in the figure. Because there is less information available at the edges of the image the algorithm tends to conservatively classify border pixels as unknown. Defocus is generally unreliable at the edge of an image, the algorithm crops final results by the larger PSF radius.

As one would expect, the trimap estimation fails as the foreground object moves beyond the depth of field, and then the optimizer cannot proceed. Given a hand-painted trimap, the optimizer degrades more gracefully and tends to estimate a blurry foreground object and matte. At some point the defocus is too severe and the foreground object becomes classified as part of the background. This is desirable, since I define background as "beyond the depth of field."

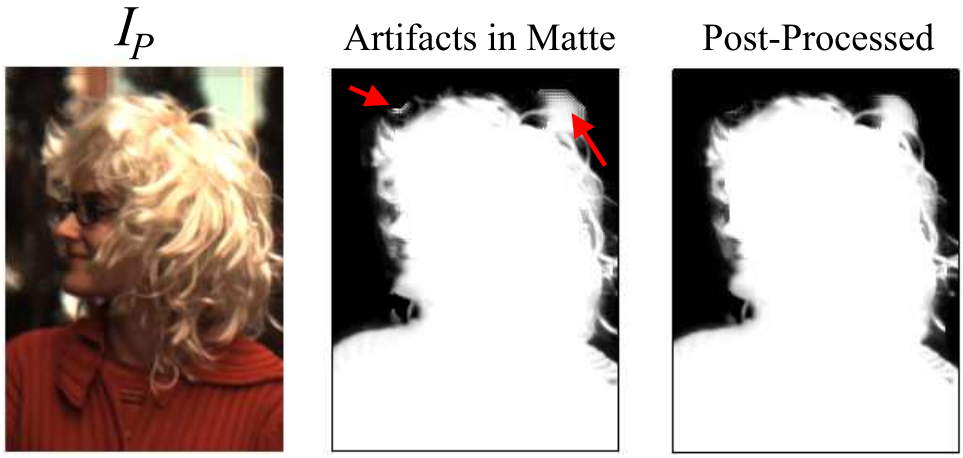


Figure 8.16: Noise on the left at the trimap border can be corrected by post-processing.

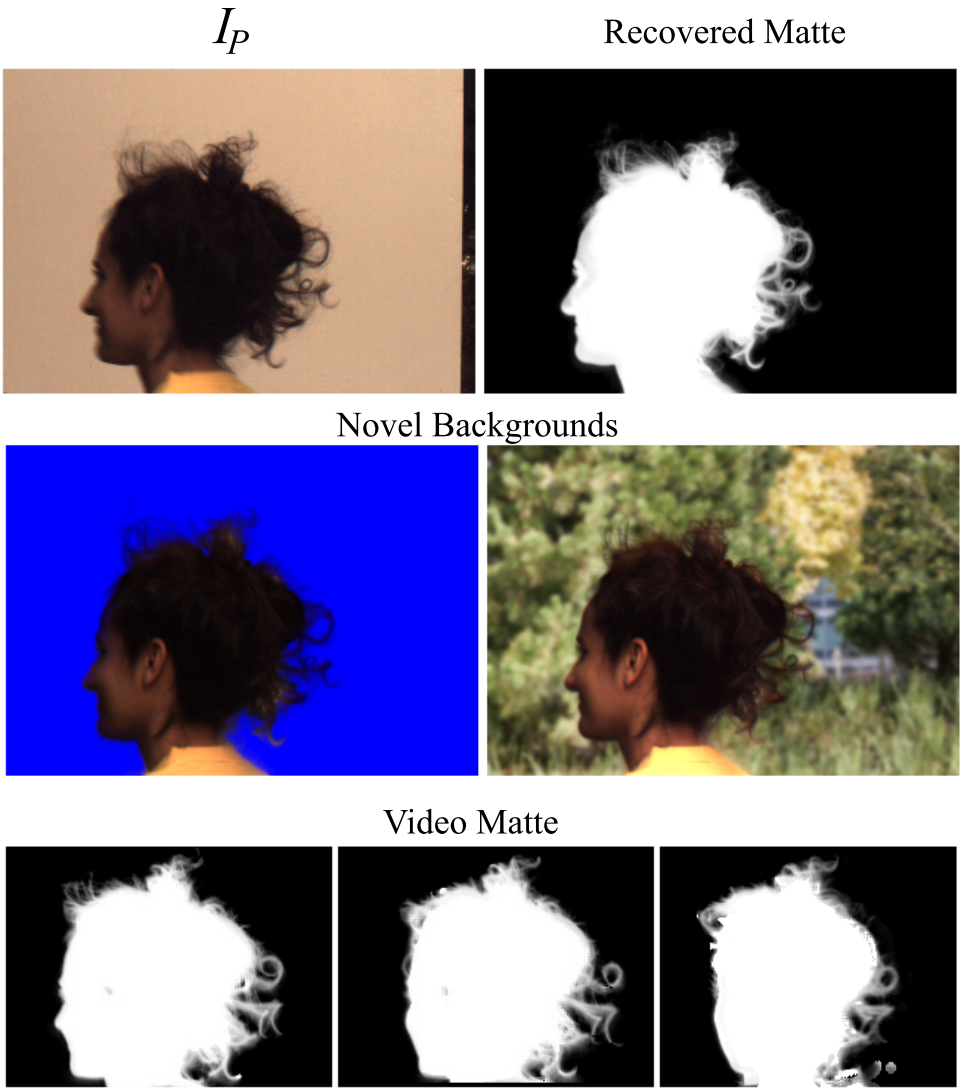


Figure 8.17: Pulling a matte from fine hair.

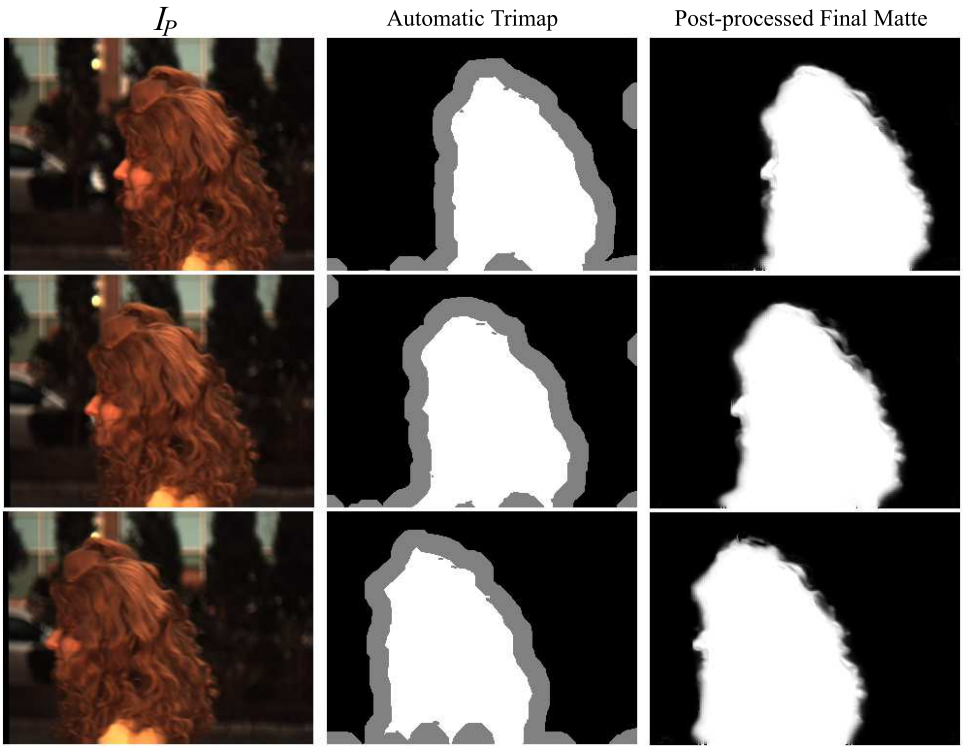


Figure 8.18: Automatic trimap-extraction in a video sequence.

Figure 8.19: Four frames of input for a matting problem.

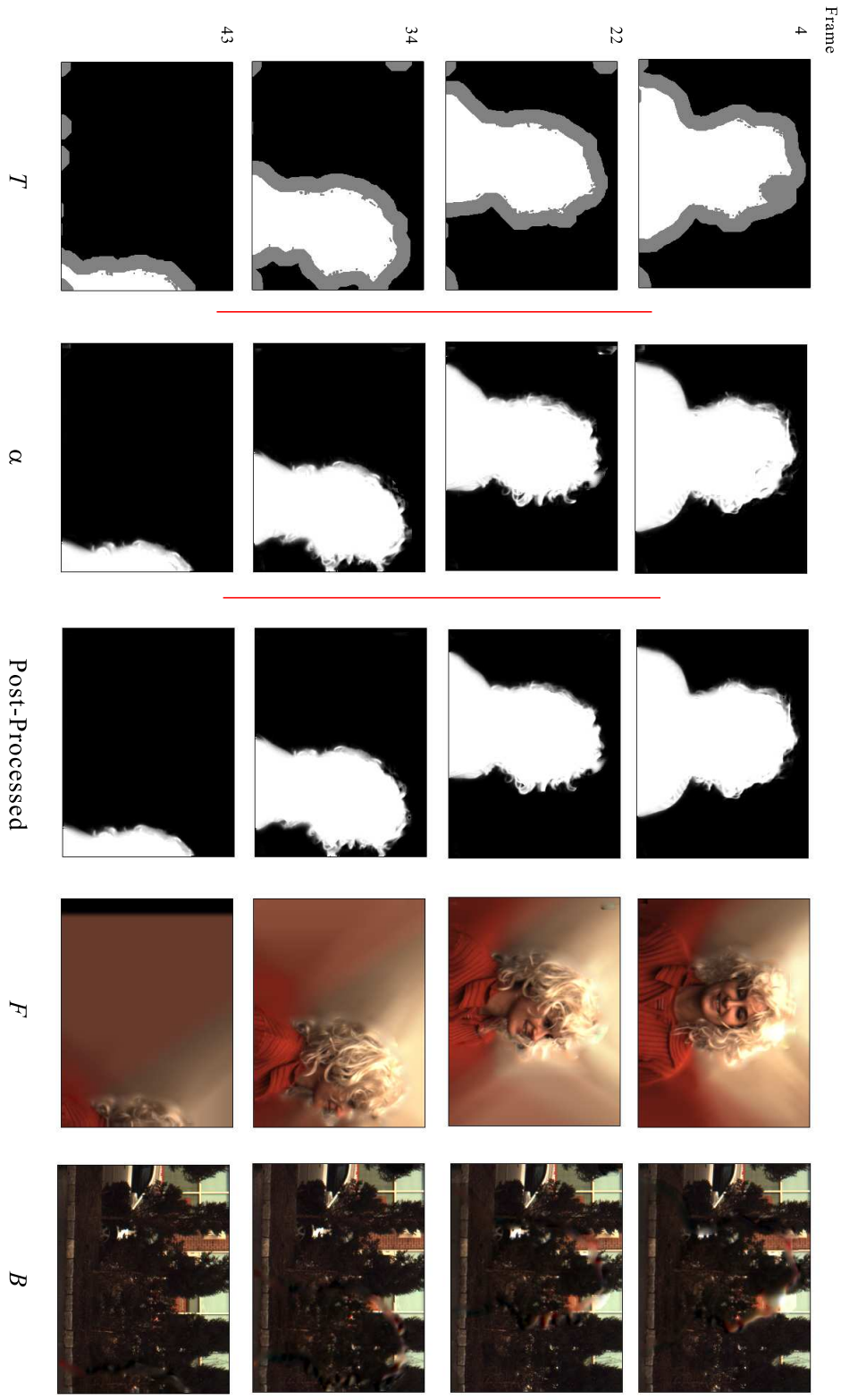


Figure 8.20: Computed intermediates and results: Trimap, matte, post-processed matte, foreground, and background.

Chapter 9

Video Synthesis

As mentioned earlier, the general SAMPL framework allows us to create lots of different applications; in this chapter I describe several of these, albeit in less detail than the defocus matting work. These applications support the claim that the capture framework is general and practical: I implemented the various video capture applications described here using different splitting trees for data acquisition, and was able to reconfigure and calibrate the system for each of these applications in a couple of hours, even when working outside of the laboratory.

9.1 Compositing

9.1.1 Novel Background

The standard application of compositing is to combine a foreground and matte with a novel background. Porter and Duff's [73] linear compositing (equation 5.1) is applied as if the new image was captured with a pinhole camera.

It is common practice to select a defocussed image for the background image so that the final composite appears natural and emphasizes the actor. Because the pinhole equation performs compositing, the composite can be thought of as an in-focus picture of a background that is inherently blurry. As discussed in chapter 3, the linear composite of a sharp foreground with a blurred background (equation 3.37) is not identical to the image that would be produced by photographing an actual scene. The discrepancy is at the borders of the foreground object, where the linear composite

produces a blurrier background than should be observed. Potmesil and Chakravarty [74] among others showed that the difference is insignificant to the human visual system, and linear compositing has been used for half a century in visual effects without perceived loss of realism due to the incorrect blur. Figure **??** shows a real photograph of an actor in front of trees (left) and a linear composite of the actor against a blurred background (right). The images do not match exactly because the background on the right was photographed in-focus and blurred for the composite, and the foreground on the right is from a rotoscoped matte and retouching in Adobe PhotoShop. The point is that both images look plausible and the highly textured background appears about equally blurry in both images. Readers experienced at finding visual artifacts will detect that the image on the right is a composite, but it is not likely that they will make that determination by examining the defocus.

Photograph Linear Composite

Figure 9.1: Comparison of real and linearly composited images.

Figure 9.2 shows a matte and foreground recovered by defocus video matting, and a new background image against which they are to be composited[1]. Note the fractional values in the matte where fine strands of hair create a translucent surface.

Figure 9.3 is the final linear composite. The inset detail (red box, zoomed in on the right) shows that the fractional alpha recovered by defocus video matting leads to the appearance of a realistic photograph, with the background appearing between hairs and blended smoothly into the hair where there is partial coverage.

[1] All novel background images are photographs by Shriram Krishnamurthi used with permission.

Figure 9.2: Inputs for the compositing process.

Figure 9.3: Result and detail from the compositing process.

9.1.2 Self-Compositing

The traditional application of compositing chooses the novel background be a completely new image. There, the matte is interpreted as partial coverage. I propose considering the matte pulled by defocus to be a selection region on the input multi-parameter video. With the notion of selection-by-matting, one can then perform video edits separately on the foreground and background. The defocus matting approach reconstructs the background as well as the foreground, so these edits can include operations that reveal a small area of background.

The simplest edit is zoom or translation of the foreground relative to the background. Figure 9.4 shows a matte and foreground pulled by defocus matting that have been manually translated and zoomed to form variations α' and F'. The original background B is preserved. These images correspond to frame 19 from the same sequence depicted in figure 8.20.

Figure 9.5 shows the original pinhole image from frame 19 on the left and the edited composite image $C + \alpha' F' + (1 - \alpha')B$ on the right. The edited image contains the desired change of scale

Figure 9.4: Scene edited by zooming and translating the foreground.

and position for the actor. It is also sharper and less noisy than the original–this is because the original was filmed with a pinhole camera and long exposure time while the composite draws the foreground primarily from the foreground-focussed sensor and the background primarily from the background-focussed sensor.

Figure 9.5: Original and edited frame of a multi-parameter video.

9.2 High Dynamic Range

High dynamic range (HDR) image acquisition is an important technique in computer vision and computer graphics to deal with the huge variance of radiance in most scenes. There have been various solutions, from commercial HDR imaging sensors (National, Silicon Vision, SMaL Camera, Pixim), to varying the exposure settings in two or more captured frames [47, 27, **?**], to changing the amount of light that is captured using a mosaic of filters [**?**]. The solution most like the one described here is by Aggarwal and Ahuja [2, 3]. They use a pyramid mirror beam-splitter and uneven offsets to direct light unevenly to three sensors. They note that this design wastes no light and achieves

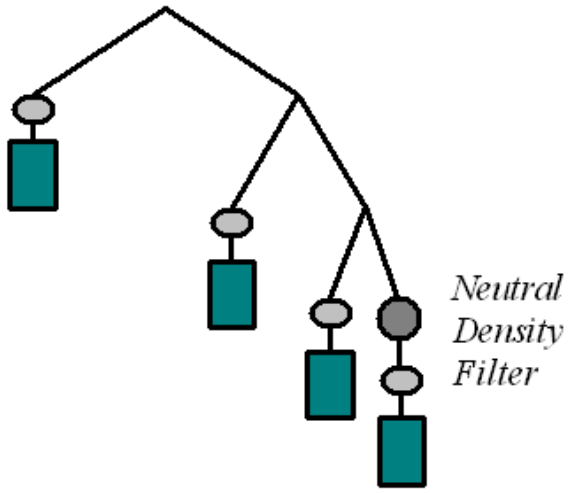


Figure 9.6: Right-Balanced Tree for HDR.

identical exposures (i.e., identical motion blur) across all cameras. I introduce a fourth sensor. More significantly, the SAMPL camera also achieves identical *apertures* across all sensors, so that the images differ solely in whole-image intensity.

An optical splitting tree has various advantages for HDR imaging. The setup naturally lends itself for HDR and only requires off-the-shelf cameras and optics. The amount of motion blur and the point spread function in each view are constant. Very little light is discarded by building an unbalanced tree of beam splitters. Figure 9.6 shows a schematic of the SAMPL camera system configured for HDR capture. For four cameras, the layout is simply the full binary tree with four beam splitters removed and a neutral-density filter inserted. Because it is a subset of the full binary tree, the HDR design easy to configure and calibrate from a pre-built full tree. More cameras can be added by extending the right subtree.

Each 50-50 beam splitter directs half the light to each of its children. The left sub-tree always terminates at a sensor. The right sub-tree recurses. Before the rightmost sensor we insert a single neutral-density 50% filter. The only light lost *by design* is 1/16 of the original at this filter. In practice, each beam splitter has also about 10% absorption. Our cameras have 10-bits of precision internally and only an 8-bit output, so we shift the gain by a factor of four from the brightest to the dimmest camera. The theoretical dynamic range for this system is thus 8192:1 for cameras that

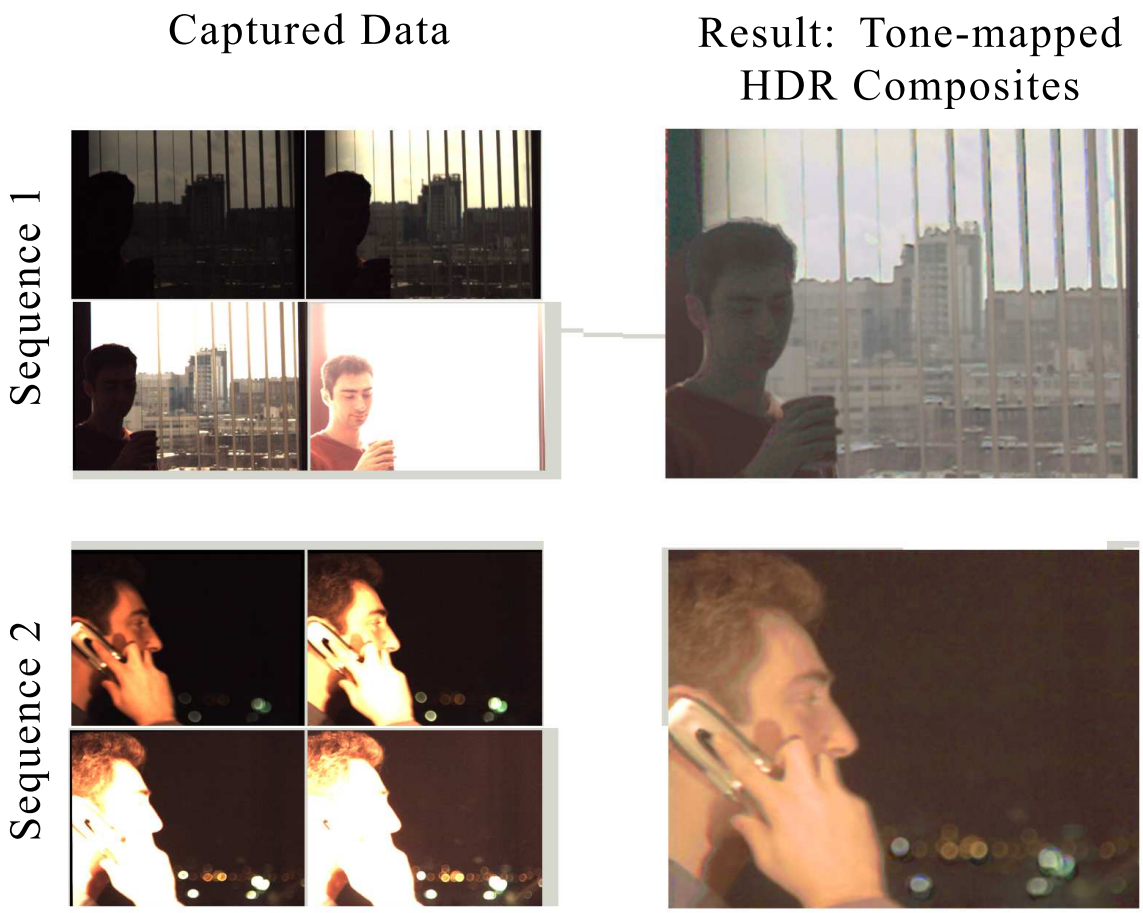


Figure 9.7: Frames from two HDR sequences.

measure linear radiance. In practice, we also vary our exposure time and apertures slightly to obtain a ratio closer to 20000:1.

For HDR, intensity and color calibration are not as important as spatial and temporal calibration because intensities will be adjusted and merged by the tone mapping process. The intensity difference between cameras can be inferred from the video streams after capture as long as some unsaturated pixels overlap between them.

Figure 9.7 shows results from two HDR experiments.The small images on the left show the data captured for a single frame by each of the four sensors. The large images on the right are the corresponding tone-mapped composites. In sequence number one (top), the actor is brightly lit and the city lights are dim in the background. In sequence number two (bottom), the actor is inside a

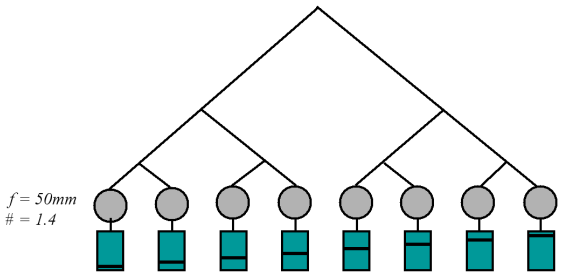


Figure 9.8: Splitting tree for capturing multi-parameter video with multiple focus samples.

dark office and the sky and city in the distance are bright. On the right are resulting tone-mapped HDR images. These combine frames from four different sensors to keep all scene elements visible and within the dynamic range of the display.

9.3 Multiple Focus and Defocus

Images focused at multiple depths can be used to recover depth information [62, 17, 92] and to form images with an infinite [83] or discontinuous [95] depth of field. Many systems (e.g., [62]) split the view behind the lens. Splitting in front of the lens allows us not only vary the location but the depth of the field by changing the aperture. This allows, for example, the use of a pinhole camera as well as a narrow depth of field camera, which has been shown to have applications in matting [**?**].

The SAMPL camera captures images with varying focus depths like those in figure 9.9 using the tree in figure 9.8. This is the full binary tree containing with eight sensors. Each sensor is focused at a different depth, ranging hyperbolically from 20 cm from the optical center (which is only about 4 cm from the first beam splitter) to 20 m (which is effectively infinity). I open the apertures of each sensor to $f/1.4$, giving a narrow depth of field and maximizing the PSF radii.

It is critical that the cameras for the more distant depths have near-perfect physical calibration, since parallax errors will be extremely large for scenes with large depth ranges. The sensors focused at nearer depths need not be calibrated as closely for most applications. This is because the extent of the depth of field falls off as the midfield depth moves closer, so a sensor focused at 20 cm has a depth of field of a few centimeters and more distant objects will be so defocused that calibration becomes irrelevant.

I have implemented two ways of visualizing multi-focused images. The first is a virtual focus depth. By blending between the two closest focused cameras, it is possible to move focusing from capture time to edit time (see figure 9.9). A single frame captured by different sensors at the same time, where each sensor has a different focus depth.

The second is an artificially wide depth of field. For this, compute a weighted sum of each sensor's image for each frame. At each pixel, the weight of a sensor's image proportional to the local contrast (high frequency information in a small neighborhood), squared. Figure 9.9 (bottom) shows the result of fusing the images in this manner.

9.4 High Speed

High speed cameras capable of 2000 fps and higher are now commercially available. Yet producing a high speed camera by combining multiple sensors remains an interesting alternative and active area of research. Wilburn et al. [91] have build a side-by-side array of 64 cameras for high speed capture and explored the issues involved with raster shutters for multi-sensor cameras. Their solution scales easily and requires less illumination than using an optical splitting tree (we must flood the scene with light). The advantage of a splitting tree approach (at least for a moderate number of cameras) is that the sensors share an optical center. This means that they accurately capture view-dependent effects and do not suffer from occlusion between views.

There are other benefits to a high-speed solution using optical splitting trees for some applications. Because adjacent frames are captured by different sensors, the exposure time and frame rate are not linked. We can capture $8 \times 30 = 240$ fps video with an exposure time of $1/30$ s, producing smooth movement with motion blur. Even when it is desirable to lock frame rate and exposure time, a single-sensor high speed frame camera's exposure can only asymptotically approach the frame rate–it must pause to discharge and measure the sensor pixels. A multi-sensor camera can discharge one sensor while capturing with another.

The data rate from a single high speed camera is enormous. Multiple sensors naturally lend themselves to parallel processing and each camera is connected using separate FireWire cables. Using multiple sensors it is also possible to combine high-speed with filters, creating for instance a

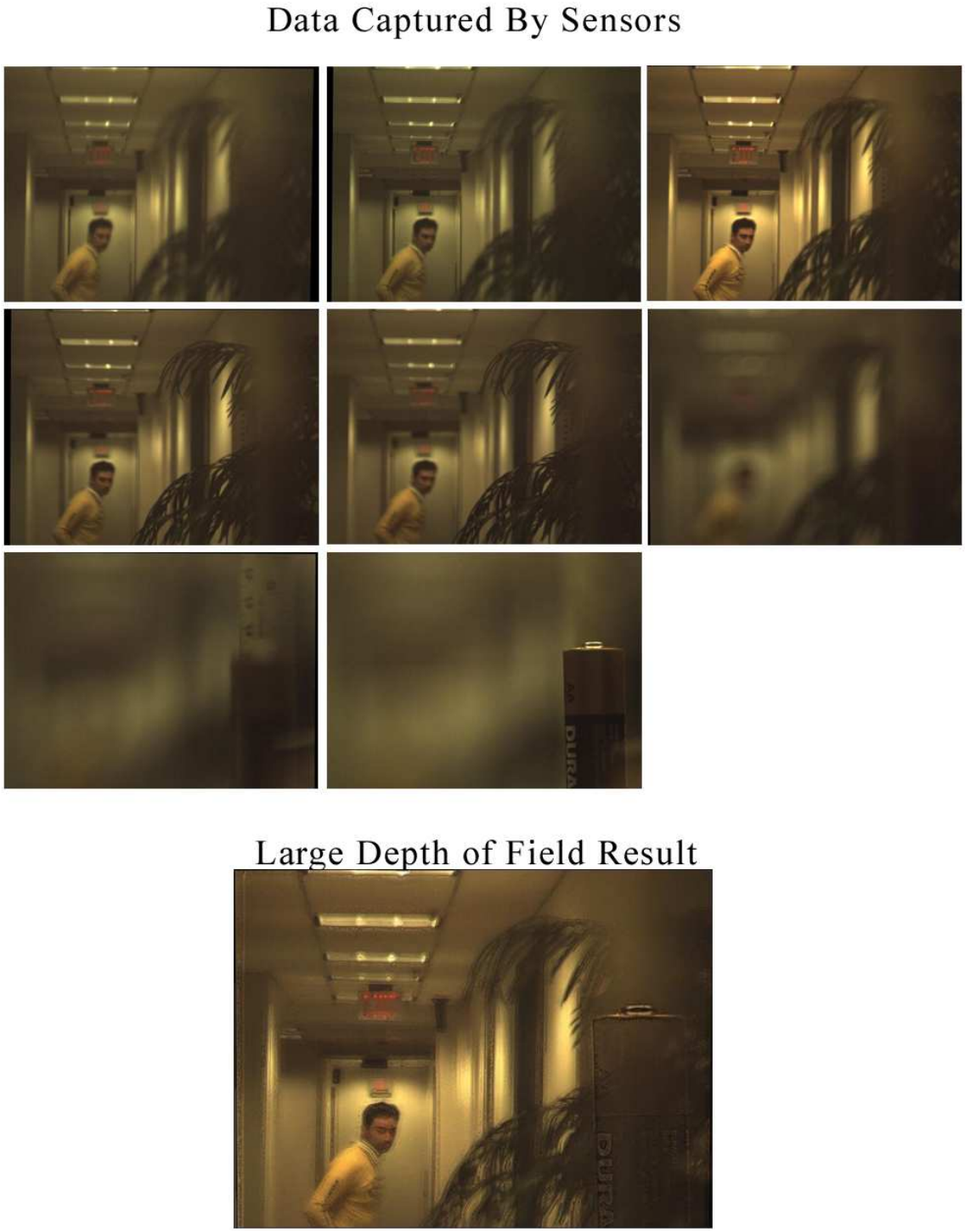


Figure 9.9: Multifocus images.

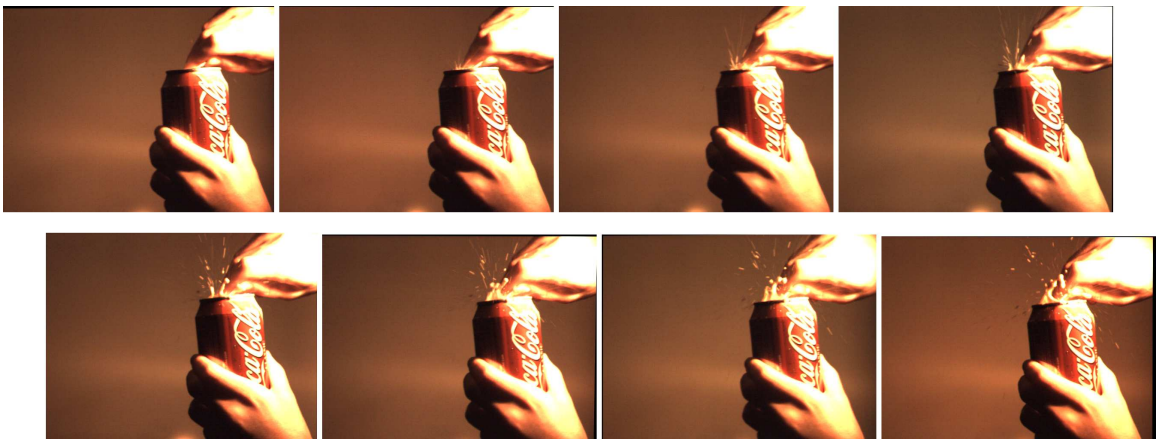


Figure 9.10: 240 fps video of a soda can opening.

combination of high-speed and multi-spectral video (see next section). Finally, for research prototyping it is attractive to have multiple sensors that can be temporarily configured for high speed without investing in a single-purpose high speed camera.

Compared to a single-sensor solution, the drawbacks of our high-speed splitting tree are the calibration problem and the loss of light. In the worst case, 7/8 of the illumination is directed at sensors that are not measuring it.

I configured the system for high speed by implementing the balanced splitting tree from Figure 6.2 with eight cameras and matching lens parameters. Figure 9.10 shows eight frames from a high speed sequence of a soda can opening. Each frame has an exposure time of $1/120$ s, and the entire sequence is captured at 240 fps. Each of the eight sequential frames shown was captured by a different sensor. The sensors have 1/120s exposure times, shorter than allowed by the frame rate for a single sensor.

9.5 Multimodal High Speed

Previous imaging devices have been constructed to increase the sampling resolution of particular parameter, like wavelength. What is interesting about a general purpose high-dimensional imaging system is that it can trade resolution between parameters by creating hybrid trees with many kinds of filters. Hybrids are more application oriented than imaging systems that undersample all other

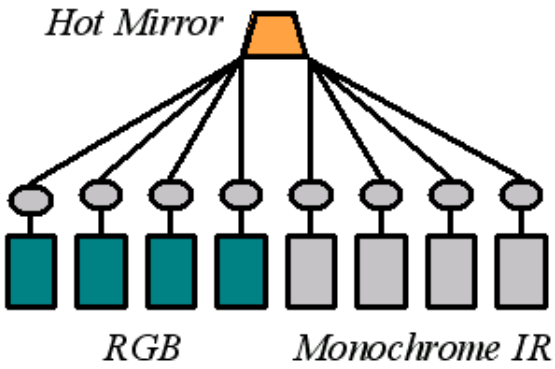


Figure 9.11: High-speed + IR Splitting Tree.

parameters in order to capture one at high fidelity.

Put another way, HDR, high speed, etc. are sampling strategies. They are useful for building high-level applications, like surveillance in an HDR range environment. It is natural to build hybrid sampling strategies, which are easy to express and experiment with within our splitting tree framework.

We use configuration shown in Figure 9.11 to capture high-speed visible light and IR video. A hot-mirror directs IR down the right sub-tree and visible light down the left sub-tree. Each subtree has four cameras with temporal phase offsets, so the entire system yields 120 fps video with four color channels. Figure 9.12 shows a few frames from a sequence in which a person catches a tumbling remote control, points it at the camera, and presses the power button. Note the tumbling remote control. The remote control's IR beam is not visible on the color sensors, but appears clearly when imaged by set of IR sensors. Because the configuration captures four color channels at 120 fps, the high-frequency binary IR pattern transmitted by the remote control is accurately recorded, as is the fast tumbling motion at the beginning of the sequence.

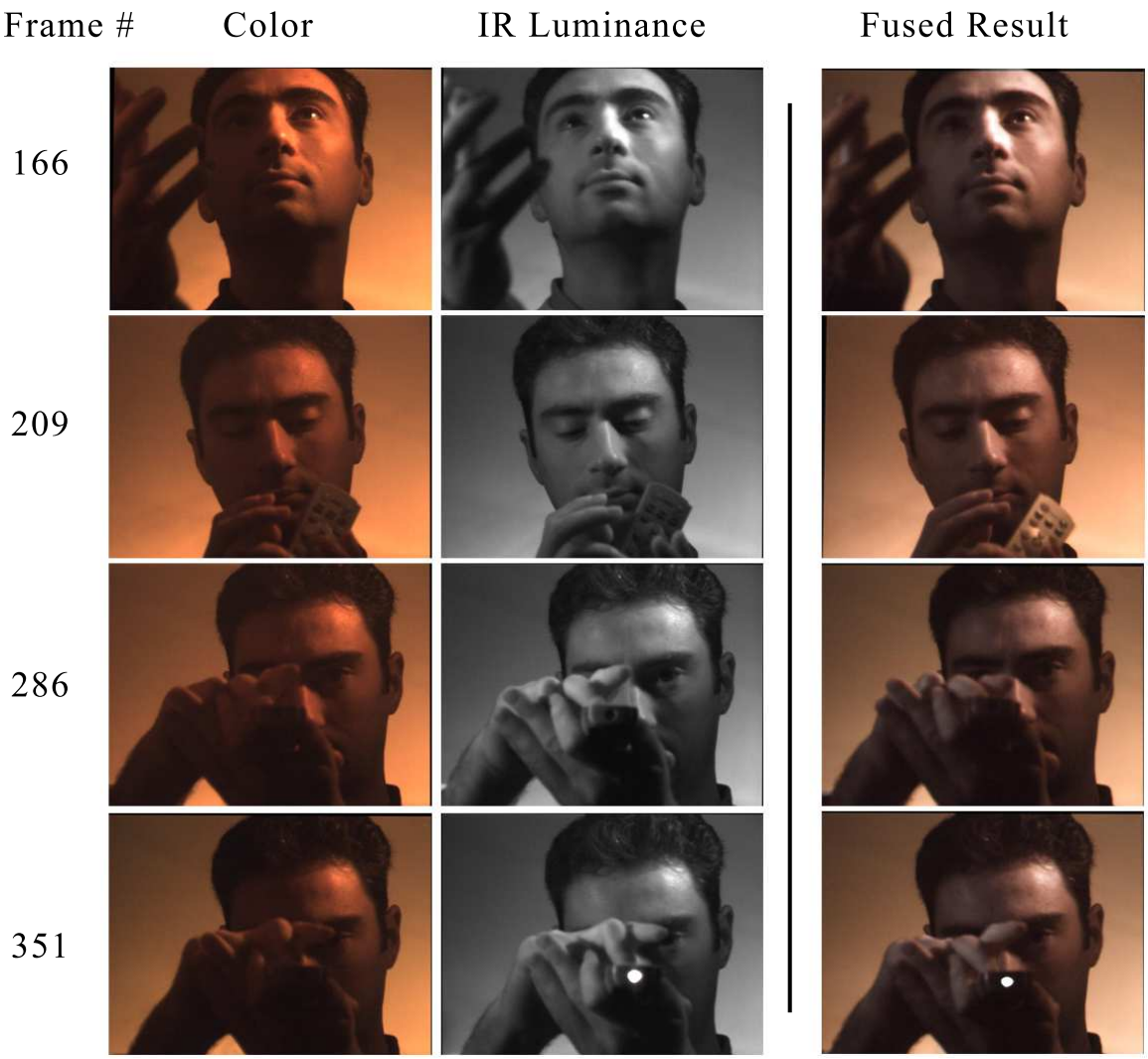


Figure 9.12: High-speed + IR Results.

Chapter 10

Discussion

This chapter begins with a recap of the contributions of this thesis. It then discusses of the broader benefits and limitations of the work. As with any research, every solved problem gives birth to new ideas on how to extend the mathematics and technology. I finish with a discussion of future work related to this project and my own continuing projects in the field.

10.1 Summary and Conclusion

This thesis introduced a multi-parameter video as new way of thinking about richly sampled image data. Multi-parameter video generalizes previous work and promotes consideration of the sampling and precision issues behind multi-view camera design. Multi-parameter video can be captured with a single-axis, multi-parameter lens (SAMPL) camera system.

Chapter 3 derived the mathematics governing the image produced by a SAMPL camera, and showed how to efficiently approximate the image captured for the particularly interesting case of a scene containing two planes of constant depth.

Chapter 6 introduced optical splitting trees as a framework to describe the important characteristics of SAMPL camera configurations. Guided by this body of theory, I built a physical SAMPL camera and captured many multi-parameter video sequences. This camera incorporates many practical design elements. It uses custom hardware to allow close physical calibration, implements a layout well-suited to experimentation, and is self-powered, physically robust, and mobile to allow that experimentation to occur outside the walls of a laboratory.

The later chapters described applications of the video sequences. The matting chapter presented two novel solutions to the classic computer graphics matting problem of separating foreground and background objects within an image. The video synthesis chapter demonstrated that the framework and SAMPL camera are a general solution for imaging problems. It reports results using the same capture system for several classic vision applications: high-dynamic range video, high-speed video, and multi-modal video; and reports on high-speed multi-modal video as an example of the ease of constructing experimental hybrids using the system.

The SAMPL camera and theory are a general platform for computer vision and computer graphics research. I have demonstrated their generality in my own research on novel computer graphics algorithms and by reproducing previous results. Parts of this work have already been picked up by other research groups. Most of my thesis research was performed at the Mitsubishi Electric Research Laboratory with Hanspeter Pfister and Wojciech Matusik. I've since collaborated with Anthony Vetro's group at the same company which has captured video sequences for the MPEG working group on multi-view video [85] using a variation on the SAMPL camera. The frameless rendering ideas (e.g., Dayal et al. [25]) of David Luebke's group at the University of Virginia are 'synthesis by sampling' in the same way that multi-parameter video is 'analysis by sampling'. The approaches are complimentary; they are now working with the data from this thesis. A final data point on generality: Frédo Durand is one of my co-authors from the SIGGRAPH matting paper. He plans to continue using the software I developed with his graphics group at MIT.

10.2 Discussion

The splitting tree and multi-parameter framework provides a forum for discussion of what is really important in rich imaging. During camera design, there must be decisions to about how to partition the incoming light into samples of several parameters. I believe that the right way to design multi-view cameras is to make that decision consciously by treating incident light as a scarce resource and considering all camera parameters as equally interesting potential samples. An output multi-parameter video is the goal and the splitting tree is a way of achieving that goal. The diagramming notation for splitting trees keeps the designer's focus on distributing light precision.

With this goal of on preserving light precision, consider limitation of light loss in the defocus video matting system. As mentioned previously in the related work section, Doug Roble tells me that filmmakers do not like putting beam splitters in front of cameras because it reduces the light that reaches their cameras [78]. The matting system is intended for film and television production, and it uses two beam splitters. Will the industry reject it? There are many factors at work. Although unpopular for film, beam splitters are widely used in television production–TelePrompTers place a monitor and the camera behind a giant beam splitter. The matting system uses beam splitters to divide light, but it also uses all of the light captured by the cameras to form the final image. Thus the precision directed away from the foreground camera and towards the background and pinhole cameras still affects the foreground image. In the matting results, several of the computed F images are sharper and less noisy than the input images. These are the mitigating factors. The limiting one is that only about $\frac{1}{16}$ of the originally available light reaches the foreground camera. Is $\frac{1}{16}$ of the incident light too small a factor for film? While under indoor (office) lighting the camera produced underexposed images, working outside in direct sunlight it actually produced overexposed images. In the Boston winter, an overcast day provided the happy medium. The lighting on a film set is artificial and incredibly bright–much more like that of outdoors than an that of an office setting–so while light loss still leads to imprecision, film sets likely so much light available that the matting technique is viable.

The dynamic range problem just mentioned is another limitation of the current camera. Basler produces very expensive computer vision cameras, but they are simply not up to the job of capturing the dynamic range of a typical outdoor scene. In fact, I suspect that the limited range is an asset for traditional vision work in robotics, where saturation helps with object classification and edge detection. The Cannon Digital Rebel camera used in the defocus difference matting experiments has much greater dynamic range, but it is a still camera. The ideal camera of course provides high resolution, high dynamic range, and high speed for less than $1000. Today the choices at that price level is $640 \times 480 \times 30$fps with eight bits and twelve-bits at two frames per second. Camera prices are falling fast due to increasing consumer demand and new manufacturing technologies, so I believe that the affordable cameras will soon catch up with the splitting tree approach.

The defocus difference matting (static background) approach is more likely to see immediate application in industry than the defocus video matting one. It uses a single beam-splitter and the pinhole camera has a larger aperture than in defocus video matting, so the foreground camera receives closer to one quarter of the original light. The technique is computationally inexpensive, so it can easily run in real-time. Recall that the state of the art in production is still blue-screen matting. Defocus difference matting avoids the blue-spill problem and even allows the use of an arbitrary background, as long as the background is static and has high frequency texture. It is well-suited to replace the blue-screens for television weather reports.

Like other natural image matting methods, the defocus approaches is limited to scenes where the foreground and background are visually distinguishable. If the boundaries of an actor, say, in a white shirt against a white wall, are hard to distinguish visually in an image, no natural image matting technique can succeed. Adding wavelength samples outside the visible spectrum (e.g., near field infra-red) increases the probability that similarly- colored materials can be distinguished. Investigating such a configuration is interesting future work, as is further experimentation with other optimization methods for solving the matting problem. Because only the Jacobian expressions are based on our choice of camera parameters, the defocus matting framework can be naturally extended to incorporate other camera parameters like variable focal length, high dynamic range, and stereo video.

Technologies that are today too exotic and limited (e.g., holographic aperture, active optics) for the purposes of this thesis offer great promise that in the future it will be able to build systems that are both compact and can virtualize most image parameters. Thus the limitations of the prototype camera are not necessarily impediments to the application in future devices with smaller form factors of the multi-parameter video processing algorithms developed with it.

10.3 Future Work

On the theory side, the next step is a methodology for achieving formal design considerations in a SAMPL camera. I envision a design process with a set of constraints as input and an optical splitting tree topology as output. The constraints describe the relative weights of different parameters. The

process of finding an optimal splitting tree is one that balances light loss, variance, component price, tree footprint, and the input constraints. If the design process can be formalized, then it can be automated. For complex constraint sets, an optimization algorithm more likely to produce light-efficient splitting tree designs than a person is. I'm working right now with Shree Nayar, John Hughes, Hanspeter Pfister, and Wojciech Matusik, to create such an optimizer. If we are successful, then future vision researchers will be able to enter their design goals and budget into a computer that will compute a camera design to meet their goals. The design should take the form of a list of part numbers from standard optical catalogs like Edmund's and a blueprint showing how to assemble the system. I expect that the catalog manufacturers will appreciate automated design as well!

It is hard to calibrate large numbers of cameras sharing a view through half-silvered mirrors. This is because there are many degrees of freedom for each component and the optical path is long compared to the size of the components. I used several calibration and correction techniques: the rifle sighting approach that uses patterns at different depths, laser pinchers for visualizing the optical path, rapid toggling between cameras, difference images, and user feature-based software correction of registration errors. I believe there is more work to be done on the calibration problem for large numbers of single-axis views.

In addition to solving calibration, it is possible to simply avoid or ignore the problem. The generic SAMPL camera is good for prototyping camera design in the laboratory because it is reconfigurable, but for deploying solutions it is desirable to freeze the design and build the splitting tree with the fewest possible components and smallest possible footprint.

For defocus difference matting, I am currently exploring matting-only cameras with Wojciech Matusik at MERL. One way of creating multiple views that I have not previously discussed is time multiplexing. The idea is to implement a splitting tree using a single imager and different filters, where each filter is used for a single frame and then the next is swapped in. This continues until all filters have been used, and then the cycle repeats. Kang et al. [47] recently demonstrated HDR video by this approach, using different exposures for alternate frames and fusing the results. For defocus difference matting, all camera parameters are constant between I_F and I_P, except for aperture. To obtain the temporal sampling fidelity of 30 fps video, time-multiplexed DDM requires 60 fps and

some motion compensation. The challenge is changing aperture size that fast. Computer driven irises take about $\frac{1}{4}$ second to adjust, and we will have to change the iris from fuly-open to fully-closed in less than $\frac{1}{120}$ of a second in order to avoid cutting into the exposure time. Shree Nayar is currently working with lensless cameras that use an LCD grid as a rapidly adjustable aperture. Wojciech and I believe that something similar may work for our camera.

Defocus video matting is very exciting as a radically new approach to the matting problem. The results are comparable to previous research techniques, but not yet good enough to replace blue-screen matting. A number of areas of future work suggest themselves.

Although DVM does not depend on high-frequency texture, it performs best when objects in the scene are textured. This makes it complementary to Bayesian and Poisson matting, which work best on low-frequency scenes. A natural next step is to combine the algorithms.

I chose the optimizer for DVM based on time and space constraints for the implementation. Strict gradient descent is known to be inferior to other algorithms with regard to convergence, however. Random restarts to avoid local minima in the error function are a good start at improving the optimizer. The larger question is how to get the convergence properties of conjugate gradient descent or monte carlo methods without incurring unacceptable memory costs.

Because it uses an optimizer, DVM has the potential to ignore the physical calibration process entirely. I am investigating allowing the position of the cameras to be a free variable in the optimization. This can be done easily by injecting an affine transformation into $C(u)$. The challenge here is making the affine transformation fast (given the complicated indexing schemes) and pushing it through the symbolic derivatives.

Temporal and spatial coherence methods and learning methods have proven powerful tools in computer vision, and they have previously been successfully applied to the matting problem. Hanspeter Pfister and I are now looking at using those tools on defocussed images to further improve DVM.

In conclusion, the field of digital video capture and processing is wide open and every conference brings exciting new results. To create the SAMPL camera and explore its applications in computational videography I collaborated with researchers from many backgrounds at Columbia,

MIT, Brown, and MERL. This kind of collaboration is the direction of future research in videography, because the problems and solutions are end-to-end ones involving capture and processing, analysis and synthesis. The significant results lie not in the areas of image processing, graphics, vision, and optics, but in the spaces for collaboration between them.

Appendix A

Gradient Computation Code

The following is C-code that computes the gradient of the error function ($\nabla Q = -\vec{E}^T \mathbf{J}$) for defocus video matting. This is the key inner loop body of the optimizer. The code follows the math and notation directly. A major challenge in implementing the algorithm is using an efficient indexing scheme for the sparse Jacobian elements and managing the associated book-keeping that results. The code is in two parts, each prepared as a MEX file for use with Matlab. The first part computes the indices of the (potentially) non-zero elements of the Jacobian. The second part

A.1 Indices of Non-zero Elements

```
/**
  @file computeJacobianIndicesImpl.c
    This is a MEX-file for MATLAB. Compile at the Matlab prompt with:

    compileIfOutOfDate('computeJacobianIndicesImpl');

    The calling syntax is:

    [J_hgind, J_FBind, J_Bconvgind, J_aind, J_aconvhind, Jrows2, Jcols2, t] = ...
          computeJacobianIndicesImpl(N, K, yxToCind, h, ...
                                     g, Omega, alphaOmegaInds, R, a, F, sz)

    See also computeJacobianIndices.m.

  Throughout the code, 'sz' is the variable 'oldsz' (i.e., the preallocation size)
  that appears in optimizeblock.m.
*/

/* mymex defines the Matrixf64/Matrixi32 structs, which allow both 1-based
 indexing through the melt (Matlab-Element) pointer and 0-based indexing
through the elt (C-Element) pointer.*/
 #include "mymex.h"

/** The function implemented by this MEX file */ static void
computeJacobianIndicesImpl(
    /* Input arguments */
```

```
const int              N,
const int              K,
const Matrixf64        yxToCind,
const Matrixf64        h,
const Matrixf64        g,
const Matrixf64        Omega,
const Matrixi32        alphaOmegaInds,
const int              R,
const Matrixf64        a,
const Matrixf64        F,
const int              sz,

/* Output arguments (must be preallocated, data referenced by them will be mutated). */
Matrixi32              J_hgind,
Matrixi32              J_FBind,
Matrixi32              J_Bconvgind,
Matrixi32              J_aind,
Matrixi32              J_aconvhind,
Matrixi32              Jrows2,
Matrixi32              Jcols2,
int*                   t_ptr ) {

int t = 0;

int i, j, k, x, y, z, L, n_a, n_F, n_B;
double vh, vg;

/* Size of F extended with a z dimension */
int FzSize[4];
/* Size of a extended with a z dimension */
int azSize[4];

FzSize[0] = F.size[0];
FzSize[1] = F.size[1];
FzSize[2] = F.size[2];
FzSize[3] = 3;

azSize[0] = a.size[0];
azSize[1] = a.size[1];
azSize[2] = 3;
azSize[3] = 1;

if (J_hgind.size[0] != sz) {
    ERROR("J_hgind_output_matrix_was_allocated_with_the_wrong_size.");
}

if (J_FBind.size[0] != sz) {
    ERROR("J_FBind_output_matrix_was_allocated_with_the_wrong_size.");
}

if (J_Bconvgind.size[0] != sz) {
    ERROR("J_Bconvgind_output_matrix_was_allocated_with_the_wrong_size.");
}

if (J_aind.size[0] != sz) {
    ERROR("J_aind_output_matrix_was_allocated_with_the_wrong_size.");
}

if (J_aconvhind.size[0] != sz) {
    ERROR("J_aconvhind_output_matrix_was_allocated_with_the_wrong_size.");
}

if (Jrows2.size[0] != sz*3) {
     printf ("sz*3 = %d, size = %d x %d x %d\n", sz*3, Jrows2.size[0], Jrows2.size[1], Jrows2.size[2]);
    ERROR("Jrows2_output_matrix_was_allocated_with_the_wrong_size.");
}
```

```
if (Jcols2.size[0] != sz*3) {
    ERROR("Jcols2 output matrix was allocated with the wrong size.");
}

/* The t index is 0-based in this code because it is in C, but all other indices
   are 1-based for Matlab routines. */

/* Iterate over unknowns */
for (n_a = 1; n_a <= N; ++n_a) {

    /* For each n_a there are seven rows with potentially
    nonzero entries in J (the rows may overlap with other
    values of n_a).
    */

    /* Pixel coordinates of u(n_a) */
    ind2sub2(Omega.size, Omega.dims, alphaOmegaInds.melt[n_a], &j, &i);

    for (L = 1; L <= 3; ++L) {

        /* The indices for F start after the indices for alpha.
           There are three colors for each pixel. */
        n_F = n_a + N   + (L-1) * N;
        n_B = n_a + 4*N + (L-1) * N;

        /* Iterate over constraints */
        for (y = j - R; y <= j + R; ++y) {
            for (x = i - R; x <= i + R; ++x) {
                for (z = 1; z <= 3; ++z) {

                    /* Values of the point spread functions at k, n.
                    We'll end up looking these up again later, but
                    we have to avoid elements outside the PSF radius.
                    */
                    vh = get3f64(&h, y - j + R + 1, x - i + R + 1, z);
                    vg = get3f64(&g, y - j + R + 1, x - i + R + 1, z);

                    /* Only operate on areas with non-zero terms */
                    if (vh != 0.0 || vg != 0.0) {

                        /* Find k such that C(y, x, L, z) = C( constraintInds [k]) = Cvec[k].
                        (k is a 1-based index) */
                        k = (int)get2f64(&yxToCind, y, x) + K * (L-1) + (3 * K) * (z - 1);

                        Jrows2.elt[t]     = k;
                        Jrows2.elt[t+sz]  = k;
                        Jrows2.elt[t+sz*2] = k;

                        Jcols2.elt[t]     = n_a;
                        Jcols2.elt[t+sz]  = n_F;
                        Jcols2.elt[t+sz*2] = n_B;

                        J_aind.elt[t]     = sub2ind2(a.size, a.dims, j, i);
                        J_FBind.elt[t]    = sub2ind3(F.size, F.dims, j, i, L);
                        J_hgind.elt[t]    = sub2ind3(h.size, h.dims, y - j + R + 1, x - i + R + 1, z);

                        /* Note that we add 3 z values to the size of F
                        and a for the convolution results against h and g. */

                        J_Bconvgind.elt[t] = sub2ind4(FzSize, 4, y, x, L, z);
                        J_aconvhind.elt[t] = sub2ind3(azSize, 3, y, x, z);

                        t = t + 1;
                    }
                }
            }
        }
```

```
        }
    }

    /* Return as a Matlab-style 1-based index.
       We went one past the end of the C-array,
       but then backed up by one. */
    *t_ptr = t;
}

/*****************************************************************************/
/*                          Interface to Matlab
*/
/*****************************************************************************/

/** Input arguments */
enum {
    N_IN = 0,
    K_IN,
    YXTOCIND_IN,
    H_IN,
    G_IN,
    OMEGA_IN,
    ALPHAOMEGAINDS_IN,
    R_IN,
    A_IN,
    F_IN,
    SZ_IN,
    NARGIN
};

/** Output arguments */
enum {
    J_HGIND_OUT = 0,
    J_FBIND_OUT,
    J_BCONVGIND_OUT,
    J_AIND_OUT,
    J_ACONVHIND_OUT,
    JROWS2_OUT,
    JCOLS2_OUT,
    T_OUT
};


void mexFunction(
    int             nlhs,
    mxArray*        plhs[],
    int             nrhs,
    const mxArray* prhs[])  {

    int sz, R, i, N, K;
    int dims[4];

    /** This code miscompiles on WinXP unless we extract the input
        arguments before the function call. */

    Matrixi32 alphaOmegaInds, Jrows2, Jcols2;
    Matrixf64 h,g,Omega,xyToCind,a,F;


    /* Check for proper number of arguments. */
    if (nrhs != NARGIN) {
        ERROR("Not enough input arguments.");
    }

    sz = scalarInteger(prhs[SZ_IN]);
    R = scalarInteger(prhs[R_IN]);
```

```
    /* Allocate output arguments, which are int32 vectors of indices */
    dims[0] = sz; dims[1] = 1; dims[2] = 1; dims[3] = 1;
    plhs[J_HGIND_OUT]      = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);
    plhs[J_FBIND_OUT]      = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);
    plhs[J_BCONVGIND_OUT] = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);
    plhs[J_AIND_OUT]       = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);
    plhs[J_ACONVHIND_OUT] = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);

    dims[0] = sz * 3;
    plhs[JROWS2_OUT]       = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);
    plhs[JCOLS2_OUT]       = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);

    dims[0] = 1;
    plhs[T_OUT]            = mxCreateNumericArray(2, dims, mxINT32_CLASS, mxREAL);


    N = scalarInteger (prhs[N_IN]);
    K = scalarInteger (prhs[K_IN]);
    alphaOmegaInds = intMatrix(prhs[ALPHAOMEGAINDS_IN]);
    h = doubleMatrix(prhs[H_IN]);
    g = doubleMatrix(prhs[G_IN]);
    Omega = doubleMatrix(prhs[OMEGA_IN]);
    xyToCind = doubleMatrix(prhs[YXTOCIND_IN]);
    F = doubleMatrix(prhs[F_IN]);
    a = doubleMatrix(prhs[A_IN]);

    Jrows2 = intMatrix(plhs[JROWS2_OUT]);
    Jcols2 = intMatrix(plhs[JCOLS2_OUT]);

    /* Invoke the actual function */
    computeJacobianIndicesImpl(
        /* Input arguments */
        N,
        K,
        xyToCind,
        h,
        g,
        Omega,
        alphaOmegaInds,
        R,
        a,
        F,
        sz,

        /* Output arguments */
        intMatrix(plhs[J_HGIND_OUT]),
        intMatrix(plhs[J_FBIND_OUT]),
        intMatrix(plhs[J_BCONVGIND_OUT]),
        intMatrix(plhs[J_AIND_OUT]),
        intMatrix(plhs[J_ACONVHIND_OUT]),
        Jrows2,
        Jcols2,
        (int*)mxGetData(plhs[T_OUT]));
}
```

A.2 Sparse Jacobian-Dense Error Vector Product

```
/**
 @file computeJacobianProductImpl.c
  This is a MEX-file for MATLAB. Compile at the Matlab prompt with:

  compileIfOutOfDate('computeJacobianProductImpl');


  See also computeJacobianProduct.m.
*/
```

```
#include "mymex.h"

double square(double x) {
    return x * x;
}

/** The xth non-zero element of the Jacobian is described by the J* arrays,
e.g., J_hgind[x]*/
void jacobianProduct(
    /* Input arguments */
    int                 N,
    int                 K,

    const Matrixi32     J_hgind,
    const Matrixi32     J_FBind,
    const Matrixi32     J_Bconvgind,
    const Matrixi32     J_aind,
    const Matrixi32     J_aconvhind,
    const Matrixi32     alphaOmegaInds,
    const Matrixi32     Jrows,
    const Matrixi32     Jcols,

    const Matrixf64     Bconvg,
    const Matrixf64     aconvh,
    const Matrixf64     h,
    const Matrixf64     g,
    const Matrixf64     a,
    const Matrixf64     F,
    const Matrixf64     B,
    const Matrixf64     u,
    const Matrixf64     E,
    const Matrixf64     FminusB,

    const Matrixf64     dadx,
    const Matrixf64     dady,
    const Matrixf64     d2adx2,
    const Matrixf64     d2ady2,
    const Matrixf64     dFdx,
    const Matrixf64     dFdy,
    const Matrixf64     d2Fdx2,
    const Matrixf64     d2Fdy2,
    const Matrixf64     dBdx,
    const Matrixf64     dBdy,
    const Matrixf64     d2Bdx2,
    const Matrixf64     d2Bdy2,

    const double        pinholeErrorWeight,
    const double        alphaErrorWeight,

    const double        aDiscriminationBias,
    const double        FBDiscriminationBias,
    const double        aGradientBias,
    const double        FGradientBias,
    const double        BGradientBias,

    /* Output arguments */
    Matrixf64           EJ) {


    int i, n, k, X, L, sz;
    double C1, C2, E_k, u_n, ratio;

    /* Intialize to zero */
    for (n = 1; n <= EJ.size[0] * EJ.size[1]; ++n) {
        EJ.melt[n] = 0.0;
    }
```

```
sz = J_hgind.size[0];

/********** Sum-squared pixel differences terms **********/

for (i = 1; i <= sz; ++i) {
    /**** dC/da = h * (F - B x g) */
    k = Jrows.melt[i];
    n = Jcols.melt[i];

    /* Decrease the scaling for the pinhole constraints */
    C1 = (k <= 3*K) ? pinholeErrorWeight : 1.0;

    EJ.melt[n] += h.melt[J_hgind.melt[i]] *
                  (F.melt[J_FBind.melt[i]] -
                    Bconvg.melt[J_Bconvgind.melt[i]]) *
                  E.melt[k] * C1;

    /**** dC/dF = a * h */
    k = Jrows.melt[i + sz];
    n = Jcols.melt[i + sz];

    C1 = (k <= 3*K) ? pinholeErrorWeight : 1.0;
    EJ.melt[n] += a.melt[J_aind.melt[i]] * h.melt[J_hgind.melt[i]] * E.melt[k] * C1;

    /**** dC/dB = g * (1 - a x h) */
    k = Jrows.melt[i + sz*2];
    n = Jcols.melt[i + sz*2];

    C1 = (k <= 3*K) ? pinholeErrorWeight : 1.0;
    EJ.melt[n] += g.melt[J_hgind.melt[i]] * (1 - aconvh.melt[J_aconvhind.melt[i]]) *
        E.melt[k] * C1;
}

/* The alpha elements receive a scaling factor. */
for (n = 1; n <= N; ++n) {
    EJ.melt[n] *= alphaErrorWeight;
}


/********** Bias terms **********/
/* All bias derivatives terms have E_k in the denominator, which cancels in the product E'J. */

ratio = (9.0 * K) / (7.0 * N);
/*
 Color discrimination Bias

   de_n/du_n = +/- 1. Note that color discrimination is a 1-e^2 term, so the derivative
   is negated relative to other terms.
 */
k = 9 * K + 1;
for (i = 1; i <= 3*N; ++i) {
    C1 = FminusB.melt[i] * FBDiscriminationBias * ratio;

    n = N + i;
    EJ.melt[n] += -C1;

    n = 4*N + i;
    EJ.melt[n] += C1;
}

/*
 Alpha coherence bias (two rows)
*/
k += 2;
for (n = 1; n <= N; ++n) {
    EJ.melt[n] += aGradientBias * ratio * (
```

```
    /* x */
        dadx.melt[alphaOmegaInds.melt[n]] *
        d2adx2.melt[alphaOmegaInds.melt[n]] +

    /* y */
        dady.melt[alphaOmegaInds.melt[n]] *
        d2ady2.melt[alphaOmegaInds.melt[n]]
        );
}

/*
 F coherence bias (six rows)
 */

/* We're going to offset the alpha (red) indices by multiples of X
   to access the green and blue channels.*/
X = F.size[0] * F.size[1];

/* Iterate over color channels */
for (L = 1; L <= 3; ++L) {
    k += 2;
    for (i = 1; i <= N; ++i) {
        n = L*N + i;
        EJ.melt[n] +=
            FGradientBias * ratio * (
            /* x */
              dFdx.melt[alphaOmegaInds.melt[i] + X*(L-1)] *
              d2Fdx2.melt[alphaOmegaInds.melt[i] + X*(L-1)] +

            /* y */
              dFdy.melt[alphaOmegaInds.melt[i] + X*(L-1)] *
              d2Fdy2.melt[alphaOmegaInds.melt[i] + X*(L-1)]
          );
    }
}

/*
 B coherence bias (two rows)
 */
/* Iterate over color channels */
for (L = 1; L <= 3; ++L) {
    k += 2;
    for (i = 1; i <= N; ++i) {
        n = (L+3)*N + i;
        EJ.melt[n] +=
            BGradientBias * ratio * (
            /* x */
              dBdx.melt[alphaOmegaInds.melt[i] + X*(L-1)] *
              d2Bdx2.melt[alphaOmegaInds.melt[i] + X*(L-1)] +

            /* y */
              dBdy.melt[alphaOmegaInds.melt[i] + X*(L-1)] *
              d2Bdy2.melt[alphaOmegaInds.melt[i] + X*(L-1)]
          );
    }
}


/*
 Alpha distribution bias (one row)
 */
k += 1;
for (n = 1; n <= N; ++n) {
    u_n = u.melt[n];
    EJ.melt[n] +=
        aDiscriminationBias * ratio *
        (u_n - square(u_n)) *
```

```
            (1.0 - 2.0 * u_n);
    }

}

/*****************************************************************************/
/*                          Interface to Matlab
*/
/*****************************************************************************/

/** Input arguments */
enum {
    N_IN = 0,
    K_IN,

    J_HGIND_IN,
    J_FBIND_IN,
    J_BCONVGIND_IN,
    J_AIND_IN,
    J_ACONVHIND_IN,
    ALPHAOMEGAINDS_IN,
    JROWS_IN,
    JCOLS_IN,

    BCONVG_IN,
    ACONVH_IN,
    H_IN,
    G_IN,
    A_IN,
    F_IN,
    B_IN,
    U_IN,
    E_IN,
    FMINUSB_IN,

    DADX_IN,
    DADY_IN,
    D2ADX2_IN,
    D2ADY2_IN,
    DFDX_IN,
    DFDY_IN,
    D2FDX2_IN,
    D2FDY2_IN,
    DBDX_IN,
    DBDY_IN,
    D2BDX2_IN,
    D2BDY2_IN,

    PINHOLEERRORWEIGHT_IN,
    ALPHAERRORWEIGHT_IN,

    ADISCRIMINATIONBIAS_IN,
    FBDISCRIMINATIONBIAS_IN,
    AGRADIENTBIAS_IN,
    FGRADIENTBIAS_IN,
    BGRADIENTBIAS_IN,

    NARGIN
};

/** Output arguments */
enum {
    EJ_OUT = 0,

    NARGOUT
};
```

```
void mexFunction(
    int             nlhs,
    mxArray*        plhs[],
    int             nrhs,
    const mxArray* prhs[])  {

    int N, K;
    int dims[4];

    /* Check for proper number of arguments. */
    if (nrhs != NARGIN) {
        ERROR("Not enough input arguments.");
    }
    if (nlhs != NARGOUT) {
        ERROR("Not enough output arguments.");
    }

    N = scalarInteger (prhs[N_IN]);
    K = scalarInteger (prhs[K_IN]);

    /* Allocate output arguments. */
    dims[0] = 1; dims[1] = 7*N; dims[2] = 1; dims[3] = 1;

    plhs[EJ_OUT] = mxCreateNumericArray(2, dims, mxDOUBLE_CLASS, mxREAL);

    /* Invoke the actual function */

    jacobianProduct(
        N,
        K,

        intMatrix(prhs[J_HGIND_IN]),
        intMatrix(prhs[J_FBIND_IN]),
        intMatrix(prhs[J_BCONVGIND_IN]),
        intMatrix(prhs[J_AIND_IN]),
        intMatrix(prhs[J_ACONVHIND_IN]),
        intMatrix(prhs[ALPHAOMEGAINDS_IN]),
        intMatrix(prhs[JROWS_IN]),
        intMatrix(prhs[JCOLS_IN]),

        doubleMatrix(prhs[BCONVG_IN]),
        doubleMatrix(prhs[ACONVH_IN]),
        doubleMatrix(prhs[H_IN]),
        doubleMatrix(prhs[G_IN]),
        doubleMatrix(prhs[A_IN]),
        doubleMatrix(prhs[F_IN]),
        doubleMatrix(prhs[B_IN]),
        doubleMatrix(prhs[U_IN]),
        doubleMatrix(prhs[E_IN]),
        doubleMatrix(prhs[FMINUSB_IN]),

        doubleMatrix(prhs[DADX_IN]),
        doubleMatrix(prhs[DADY_IN]),
        doubleMatrix(prhs[D2ADX2_IN]),
        doubleMatrix(prhs[D2ADY2_IN]),
        doubleMatrix(prhs[DFDX_IN]),
        doubleMatrix(prhs[DFDY_IN]),
        doubleMatrix(prhs[D2FDX2_IN]),
        doubleMatrix(prhs[D2FDY2_IN]),
        doubleMatrix(prhs[DBDX_IN]),
        doubleMatrix(prhs[DBDY_IN]),
        doubleMatrix(prhs[D2BDX2_IN]),
        doubleMatrix(prhs[D2BDY2_IN]),

        scalarDouble(prhs[PINHOLEERRORWEIGHT_IN]),
        scalarDouble(prhs[ALPHAERRORWEIGHT_IN]),
```

```
        scalarDouble(prhs[ADISCRIMINATIONBIAS_IN]),
        scalarDouble(prhs[FBDISCRIMINATIONBIAS_IN]),
        scalarDouble(prhs[AGRADIENTBIAS_IN]),
        scalarDouble(prhs[FGRADIENTBIAS_IN]),
        scalarDouble(prhs[BGRADIENTBIAS_IN]),

        doubleMatrix(plhs[EJ_OUT]));
}
```

Appendix B

Camera Mount Specifications

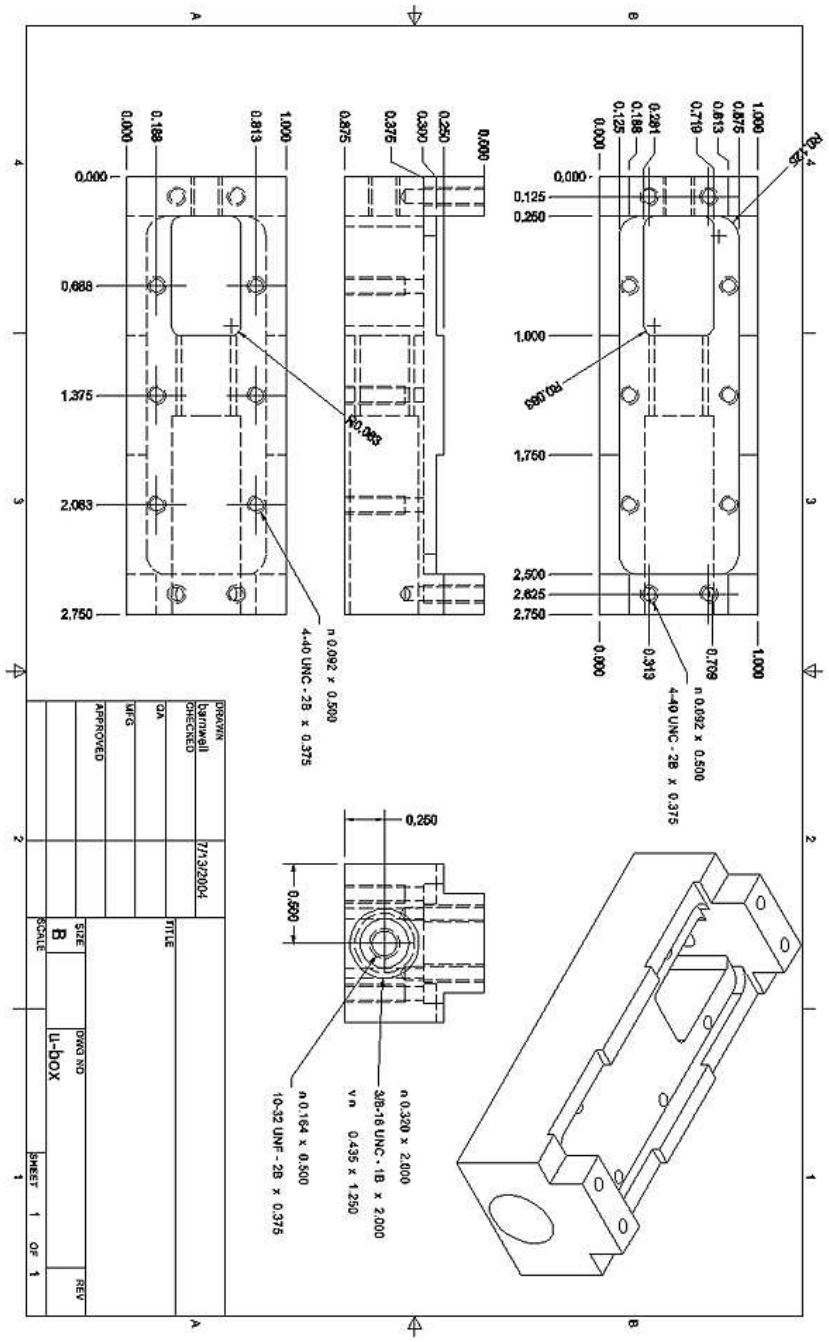


Figure B.1: Base of the sensor mount stage. A thin slide translates along one axis inside the U-gouge of the mount. A T-bar bolted to the slide rotates about one axis. The translation spring mounts in the large hole visible at one end of the long axis; the translation screw mounts in the small hole on the opposite end.

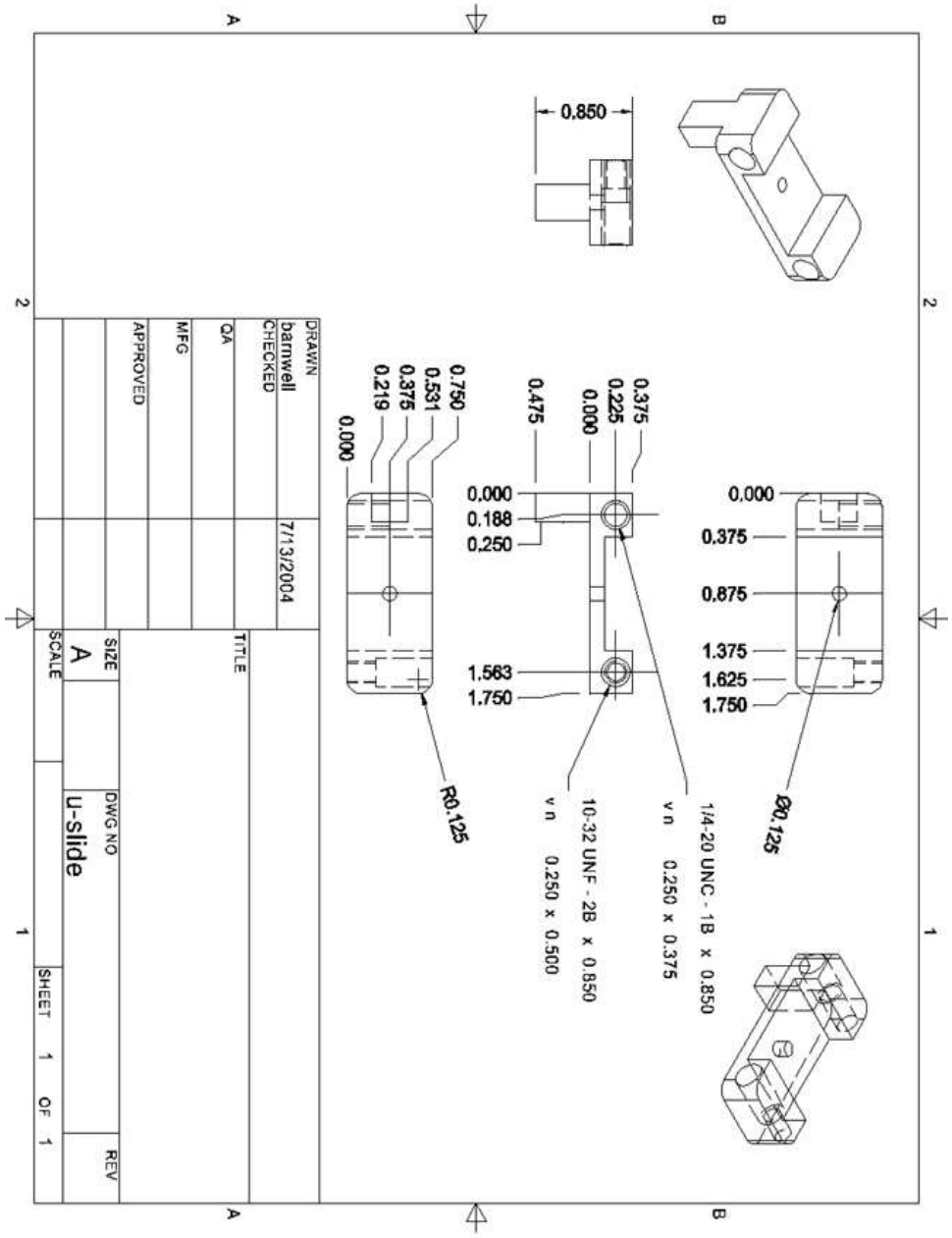


Figure B.2: Slide that rides inside the U-gouge of a sensor mount stage base. Note the cube tab that hangs below the body of the slide. The translation thumbscrew presses against this tab on the side where the tab is flush with the slide. A strong spring opposes the tab on the opposite side. The two holes running across the short axis of the slide are for the rotation thumbscrew and the spring that opposes it. The rotation spring is weak compared to the translation one because it must be shorter.

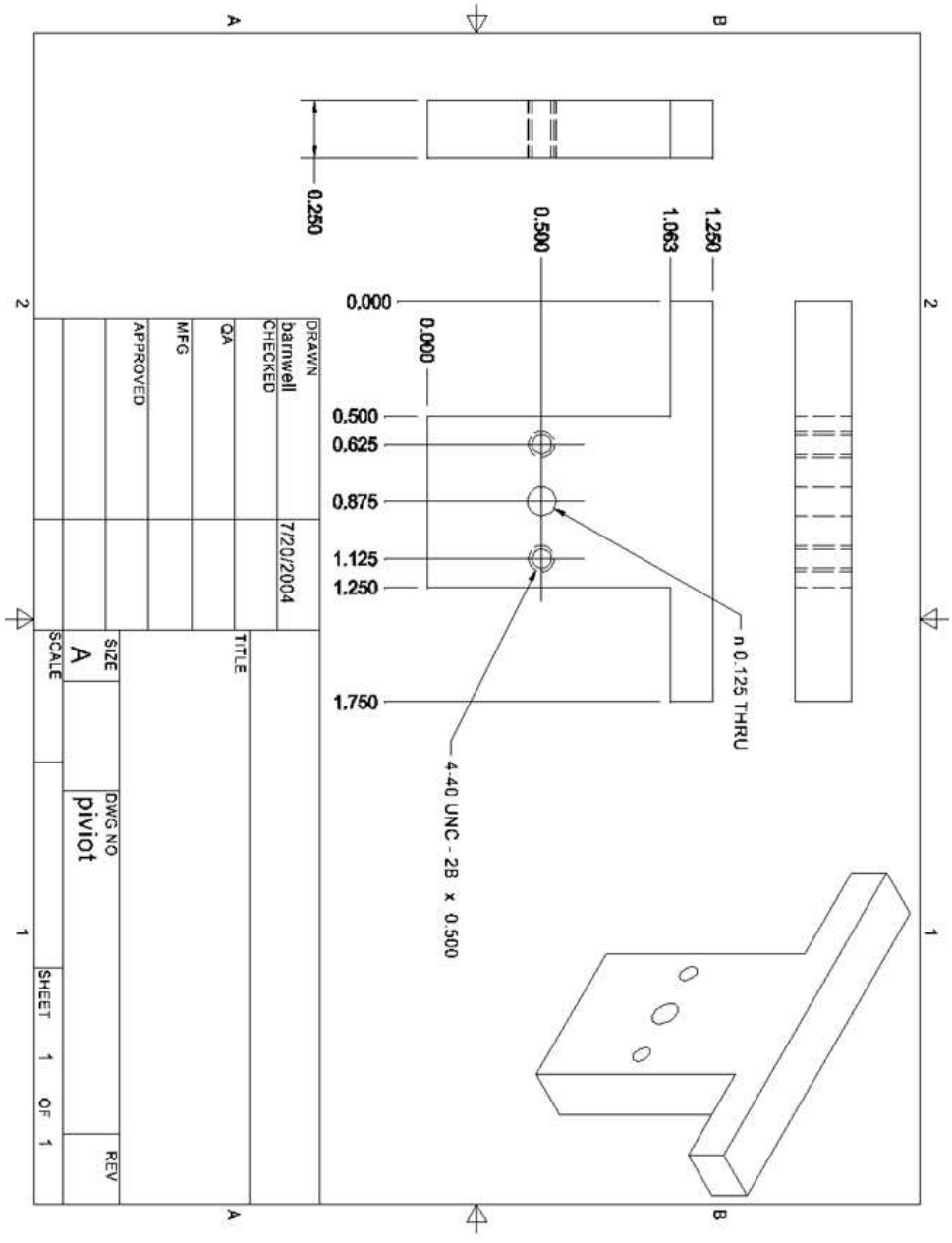


Figure B.3: T-plate that bolts to the slide in a sensor mount stage. The L-bracket for the next stage is screws into the top of the T-plate. The rotation screw and spring press against opposing arms of the T.

Bibliography

[1] Color correction for image sensors. Technical Report MTDPS-0534-2, October 2003. http://www.kodak.com/global/plugins/acrobat/en/digital/ccd/applicationNotes/ColorCorrectionforImageSe

[2] M. Aggarwal and N. Ahuja. Split aperture imaging for high dynamic range. In *ICCV01*, pages II: 10–17, 2001.

[3] M. Aggarwal and N. Ahuja. Split aperture imaging for high dynamic range. *IJCV*, 58(1):7–17, June 2004.

[4] N. E. Apostoloff and A. W. Fitzgibbon. Bayesian video matting using learnt image priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 407–414, 2004.

[5] N. Asada, H. Fujiwara, and T. Matsuyama. Seeing behind the scene: analysis of photometric properties of occluding edges by the reversed projection blurring model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(2):155–67, 1998.

[6] Naoki Asada, Hisanaga Fujiwara, and Takashi Matsuyama. Edge and depth from focus. *Int. J. Comput. Vision*, 26(2):153–163, 1998.

[7] M. Bajcsy, A. S. Zibrov, and M. D. Lukin. Stationary pulses of light in an atomic medium. *Nature*, (426):638–641, December 2003.

[8] Bryce Bayer. Color imaging array, July 1976. United States Patent 3,971,065.

[9] M. Ben-Ezra and S.K. Nayar. Jitter camera: High resolution video from a low resolution detector. In *IEEE CVPR*, pages 135–142, June 2004.

[10] Arie Berman, Arpag Dadourian, and Paul Vlahos. Method for removing from an image the background surrounding a selected object, 2000. U.S. Patent 6,134,346.

[11] Arie Berman, Paul Vlahos, and Arpag Dadourian. Comprehensive method for removing from an image the background surrounding a selected object, 2000. U.S. Patent 6,134,345.

[12] S. S. Bhasin and S. Chaudhuri. Depth from defocus in presence of partial self occlusion. *Proceedingsof the International Conference on Computer Vision*, 1(2):488–93, 2001.

[13] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr. Interactive image segmentation using an adaptive gmmrf model. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2004.

[14] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schroder. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. *ACM Trans. on Graphics*, 22(3):917–924, 2003.

[15] Peter J. Burt and Edward H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2(4):217–236, 1983.

[16] Xiaochun Cao and Mubarak Shah. Creating realistic shadows of composited objects. Jan 2005.

[17] S. Chaudhuri and A.N. Rajagopalan. *Depth from Defocus: A Real Aperture Imaging Approach*. Springer-Verlag, 1998.

[18] Yung-Yu Chuang. New models and methods for matting and compositing, 2004. Ph.D. Thesis, University of Washington.

[19] Yung-Yu Chuang, Aseem Agarwala, Brian Curless, David H. Salesin, and Richard Szeliski. Video matting of complex scenes. *ACM Trans. on Graphics*, 21(3):243–248, July 2002.

[20] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271. IEEE Computer Society, December 2001.

[21] Yung-Yu Chuang, Dan B Goldman, Brian Curless, David H. Salesin, and Richard Szeliski. Shadow matting and compositing. *ACM Trans. Graph.*, 22(3):494–500, 2003.

[22] Yung-Yu Chuang, Douglas E. Zongker, Joel Hindorff, Brian Curless, David H. Salesin, and Richard Szeliski. Environment matting extensions: towards higher accuracy and real-time capture. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 121–130. ACM Press/Addison-Wesley Publishing Co., 2000.

[23] A. Dadourian. Method and apparatus for compositing video images (u.s. patent 5,343,252), August 1994.

[24] T.J. Darrell and K. Wohn. Depth from focus using a pyramid architecture. *PRL*, 11:787–796, 1990.

[25] Abhinav Dayal, Cliff Woolley, Ben Watson, and David Luebke. Adaptive frameless rendering. In *Eurographics Symposium on Rendering*, June 2005.

[26] Paul Debevec, Andreas Wenger, Chris Tchou, Andrew Gardner, Jamie Waese, and Tim Hawkins. A lighting reproduction approach to live-action compositing. *ACM Trans. on Graphics*, 21(3):547–556, July 2002.

[27] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378. ACM Press/Addison-Wesley Publishing Co., 1997.

[28] James H. Elder and Richard M. Goldberg. Image editing in the contour domain. *IEEE PAMI*, 23(3):291–296, 2001.

[29] J. Ens and P. Lawrence. An investigation of methods for determining depth from focus. *PAMI*, 15(2):97–108, February 1993.

[30] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 249–256. ACM Press, 2002.

[31] P. Favaro and S. Soatto. Seeing beyond occlusions (and other marvels of a finite lens aperture). In *IEEE CVPR*, pages 579–586, 2003.

[32] Sidney Fels, Eric Lee, and Kenji Mase. Techniques for interactive video cubism (poster session). In *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, pages 368–370, New York, NY, USA, 2000. ACM Press.

[33] R. Fielding. *The Technique of Special Effects Cinematography, 3rd edition*. Focal/Hastings House, 1972.

[34] Graham D. Finlayson, Steven D. Hordley, and Mark S. Drew. Removing shadows from images. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV*, pages 823–836, London, UK, 2002. Springer-Verlag.

[35] Max Fleischer. Method of producing moving picture cartoons, 1917. US Patent no. 1,242,674.

[36] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990.

[37] P. R. Gill, W. Murray, and M. H. Wright. *The Levenberg-Marquardt Method*, chapter 4.7.3, pages 136–137. Academic Press, 1981.

[38] Y. Goto, K. Matsuzaki, I. Kweon, and T. Obatake. Cmu sidewalk navigation system: a blackboard-based outdoor navigation system using sensor fusion with colored-range images. In *Proceedings of 1986 fall joint computer conference on Fall joint computer conference*, pages 105–113. IEEE Computer Society Press, 1986.

[39] P. Grossmann. Depth from focus. *PRL*, 5(1):63–69, 1987.

[40] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE PAMI*, 9(4):532–550, 1987.

[41] R. P. Harvey. Optical beam splitter and electronic high speed camera incorporating such a beam splitter. United States Patent US5734507, 1998.

[42] Euguene Hecht. *Optics Third Edition*. Addison Wesley Longman, Inc., 1998.

[43] P. Hillman, J. Hannah, and D. Renshaw. Alpha channel estimation in high resolution images and image sequences. In *Proceedings of IEEE CVPR 2001*, volume 1, pages 1063–1068. IEEE Computer Society, December 2001.

[44] E. Ikeda. Image data processing apparatus for processing combined image signals in order to extend dynamic range. U.S. Patent 5801773, September 1998.

[45] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 297–306. ACM Press/Addison-Wesley Publishing Co., 2000.

[46] B. Jahne and P. Geissler. Depth from focus with one image. In *CVPR94*, pages 713–717, 1994.

[47] Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22(3):319–325, 2003.

[48] Allison W. Klein, Peter-Pike J. Sloan, Adam Finkelstein, and Michael F. Cohen. Stylized video cubes. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 15–22, New York, NY, USA, 2002. ACM Press.

[49] K. Levenberg. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, (2):164–168, 1944.

[50] Marc Levoy, Billy Chen, Vaibhav Vaish, Mark Horowitz, Ian McDowall, and Mark Bolas. Synthetic aperture confocal imaging. *ACM Trans. Graph.*, 23(3):825–834, 2004.

[51] J. Lewis. Lifting detail from darkness. In *SIGGRAPH 2001 Sketch*, 2001.

[52] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. (11):431–441, 1963.

[53] Wojciech Matusik and Hanspeter Pfister. 3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *ACM Trans. on Graphics*, 23(3):814–824, 2004.

[54] Morgan McGuire, John F. Hughes, Wojciech Matusik, Hanspeter Pfister, Frédo Durand, and Shree Nayar. A configurable single-axis, multi-parameter lens camera. In *Symposium on Computational Photography and Video Poster*, May 2005.

[55] Morgan McGuire and Wojciech Matusik. Defocus difference matting. In *SIGGRAPH 2005 Sketch*, 2005.

[56] Morgan McGuire, Wojciech Matusik, Hanspeter Pfister, John Hughes, and Frédo Durand. Defocus video matting. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 2005.

[57] Morgan McGuire, Wojciech Matusik, Hanspeter Pfister, and Shree Nayar. Optical splitting trees for high-precision monocular imaging. *Submitted to ICCV 2005*, 2005.

[58] Y. Mishima. A software chromakeyer using polyhedric slice. In *Proceedings of NICOGRAPH 92 (Japanese)*, pages 44–52, 1992.

[59] Y. Mishima. Soft edge chroma-key generation based upon hexoctahedral color space, 1993. U.S. Patent 5,355,174.

[60] Tomoo Mitsunaga, Taku Yokoyama, and Takashi Totsuka. Autokey: human assisted key extraction. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 265–272. ACM Press, 1995.

[61] S. K. Nayar and V. Branzoi. Adaptive dynamic range imaging: Optical control of pixel exposures over space and time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1168–1175, Oct 2003.

[62] Shree K. Nayar, Masahiro Watanabe, and Minori Noguchi. Real-time focus range sensor. *IEEE PAMI*, 18(12):1186–1198, 1996.

[63] S.K. Nayar and T. Mitsunaga. High dynamic range imaging: Spatially varying pixel exposures. In *CVPR00*, pages I: 472–479, 2000.

[64] S.K. Nayar and Y. Nakagawa. Shape from focus: An effective approach for rough surfaces. In *CRA90*, pages 218–225, 1990.

[65] S.K. Nayar and S.G. Narasimhan. Assorted pixels: Multi-sampled imaging with structural models. In *ECCV02*, page IV: 636 ff., 2002.

[66] Ren Ng. Fourier slice photography. In *ACM Transactions on Graphics (SIGGRAPH 2005)*, 2005.

[67] Ren Ng, Marc Levoy, Mathieu Brédif, Gene Duval, Mark Horowitz, and Pat Hanrahan. Light field photography with a hand-held plenoptic camera. Technical Report Tech Report CSTR 2005-02, April 2005.

[68] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Verlag, 1999.

[69] Illah Nourbakhsh, David Andre, Carlo Tomasi, and Michael Genesereth. Mobile robot obstacle avoidance via depth from focus. *Robotics and Autonomous Systems*, 22:151–158, June 1997.

[70] A. P. Pentland. A new sense for depth of field. *IEEE PAMI*, 9(4):523–531, 1987.

[71] Patrick Perez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. on Graphics*, 22(3):313–318, 2003.

[72] Lena Petrović;, Brian Fujito, Lance Williams, and Adam Finkelstein. Shadows for cel animation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 511–516, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[73] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259. ACM Press, 1984.

[74] Michael Potmesil and Indranil Chakravarty. Modeling motion blur in computer-generated images. *Computer Graphics*, 17(3):389–399, July 1983.

[75] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge (UK) and New York, 2nd edition, 1992.

[76] Richard J. Qian and M. Ibrahim Sezan. Video background replacement without a blue screen. In *Proceedings of ICIP*, volume 4, pages 143–146, 1999.

[77] Ramesh Raskar, Adrian Ilie, and Jingyi Yu. Image fusion for context enhancement and video surrealism. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 85–152. ACM Press, 2004.

[78] D. Roble, September 2004. Personal communication with Doug Roble of Digital Domain.

[79] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. on Graphics*, 23(3):309–314, 2004.

[80] Mark A. Ruzon and Carlo Tomasi. Alpha estimation in natural images. In *CVPR 2000*, volume 1, pages 18–25, June 2000.

[81] Yoav Y. Schechner, Nahum Kiryati, and Ronen Basri. Separation of transparent layers using focus. *International Journal of Computer Vision*, pages 25–39, 2000.

[82] Alvy Ray Smith and James F. Blinn. Blue screen matting. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 259–268. ACM Press, 1996.

[83] Murali Subbarao. Parallel depth recovery by changing camera parameters. In *Proceedings of the Second International Conference on Computer Vision*, pages 149–155, December 1988.

[84] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. *ACM Transactions on Graphics*, pages 315 – 321, August 2004.

[85] Anthony Vetro, Morgan McGuire, Wojciech Matusik, Alexander Behrens, Jinho Lee, and Hanspeter Pfister. Multiview video test sequences from merl for the mpeg multiview working group. *ISO/IEC JTC1/SC29/WG11 Document m12077*, April 2005.

[86] Z. Vidor. An infrared self-matting process. *Society of Motion Picture and Television Engineers*, (69):425–427, June 1960.

[87] P. Vlahos. Composite photography utilizing sodium vapor illumination (u.s. patent 3,095,304), May 1958.

[88] P. Vlahos. Electronic composite photography (u.s. patent 3,595,987, July 1971.

[89] P. Vlahos. Comprehensive electronic compositing system (u.s. patent 4,100,569), July 1978.

[90] Huamin Wang and Ruigang Yang. Towards space: time light field rendering. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 125–132, New York, NY, USA, 2005. ACM Press.

[91] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Marc Levoy, and Mark Horowitz. High speed video using a dense camera array. In *Proceedings of CVPR04*, pages 294–301, June 2004.

[92] Y. Xiong and S.A. Shafer. Depth from focusing and defocusing. In *DARPA93*, page 967, 1993.

[93] Giora Yahav and Gavriel Iddan. 3dv systems' zcam. *Broadcast Engineering*, 2002.

[94] Hugh D. Young. *University Physics*. Addison-Wesley, 1992.

[95] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. on Graphics*, 23(3):600–608, 2004.

[96] Douglas E. Zongker, Dawn M. Werner, Brian Curless, and David H. Salesin. Environment matting and compositing. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 205–214. ACM Press/Addison-Wesley Publishing Co., 1999.