

Computer Vision: Algorithms and Applications

Richard Szeliski

December 23, 2008

Overview

1	Introduction	1
2	Image formation	29
3	Image processing	97
4	Feature detection and matching	207
5	Geometric alignment and calibration	309
6	Structure from motion	325
7	Dense motion estimation	335
8	Image stitching	367
9	Computational photography	407
10	Stereo correspondence	459
11	Shape and appearance modeling	481
12	Image-based rendering	491
13	Video-based rendering	499
14	Recognition	503
15	Conclusions (or better title)	513

[*Note: To make the page numbers up to date, run the “make” command, which will generate book.ovr.*

If you want leaders after chapters, enable the code at the bottom of mybook.sty. Also, if you want the overview tighter, replace chapter with section in book.ovr (use sed script in Makefile).

Appendices are currently left out. Modify Makefile to restore.]

Contents

Contents	v
Preface	xv
List of symbols	xxi
1 Introduction	1
1.1 A brief history	11
1.2 Overview	19
2 Image formation	29
2.1 Geometric image formation	32
2.1.1 Geometric primitives	32
2.1.2 2D transformations	35
2.1.3 3D transformations	39
2.1.4 3D to 2D projections	46
2.1.5 Lens distortions	58
2.2 Photometric image formation	60
2.2.1 Lighting	60
2.2.2 Reflectance and shading	61
2.2.3 Optics	68
2.3 The digital camera	73
2.3.1 Sampling and aliasing	77
2.3.2 Color	80
2.3.3 Compression	91
2.4 Exercises	92
3 Image processing	97
3.1 Local operators	99

3.1.1	Pixel transforms	101
3.1.2	Color transforms	102
3.1.3	Compositing and matting	103
3.1.4	Histogram equalization	105
3.1.5	<i>Application:</i> Tonal adjustment	109
3.2	Neighborhood operators	109
3.2.1	Linear filtering	111
3.2.2	Non-linear filtering	121
3.2.3	Morphology	126
3.2.4	Distance transforms	127
3.2.5	Connected components	129
3.3	Fourier transforms	131
3.3.1	Wiener filtering	139
3.3.2	<i>Application:</i> Sharpening, blur, and noise removal	143
3.4	Pyramids and wavelets	144
3.4.1	Interpolation and decimation	145
3.4.2	Multi-resolution representations	151
3.4.3	Wavelets	156
3.4.4	<i>Application:</i> Image blending	162
3.5	Geometric transformations	164
3.5.1	Parametric transformations	165
3.5.2	Mesh-based warping	171
3.5.3	<i>Application:</i> Feature-based morphing	174
3.6	Global optimization	175
3.6.1	Regularization	176
3.6.2	Markov Random Fields	181
3.6.3	<i>Application:</i> Image restoration	194
3.7	Exercises	194
4	Feature detection and matching	207
4.1	Points	209
4.1.1	Feature detectors	212
4.1.2	Feature descriptors	224
4.1.3	Feature matching	228
4.1.4	Feature tracking	237
4.1.5	<i>Application:</i> Performance-driven animation	239
4.2	Edges	240

4.2.1	Edge detection	241
4.2.2	Edge linking	249
4.2.3	<i>Application:</i> Edge editing and enhancement	253
4.3	Lines	255
4.3.1	Successive approximation	255
4.3.2	Hough transforms	256
4.3.3	Vanishing points	261
4.3.4	<i>Application:</i> Rectangle detection	263
4.4	Active contours	264
4.4.1	Snakes	265
4.4.2	Scissors	275
4.4.3	Level Sets	276
4.4.4	<i>Application:</i> Contour tracking and rotoscoping	278
4.5	Region segmentation	278
4.5.1	Split and merge	280
4.5.2	Mean shift and mode finding	284
4.5.3	Normalized cuts	291
4.5.4	Graph cuts and energy-based methods	296
4.5.5	<i>Application:</i> Medical image segmentation	299
4.6	Exercises	300
5	Geometric alignment and calibration	309
5.1	Feature-based alignment	311
5.1.1	Least squares	311
5.1.2	<i>Application:</i> Panography	312
5.1.3	Iterative algorithms	312
5.1.4	<i>Application:</i> Photo stabilization	314
5.1.5	Robust least squares and RANSAC	314
5.1.6	3D alignment	316
5.1.7	Uncertainty modeling	317
5.2	Pose estimation (extrinsic calibration)	317
5.2.1	Linear algorithms	317
5.2.2	Iterative algorithms	318
5.2.3	<i>Application:</i> Videomouse	318
5.3	Geometric (intrinsic) calibration	318
5.3.1	Calibration patterns	319
5.3.2	Vanishing points and lines	320

5.3.3	<i>Application:</i> Single View Metrology	320
5.3.4	Rotational motion	320
5.3.5	Radial distortion	320
5.4	Exercises	321
6	Structure from motion	325
6.1	Triangulation	327
6.2	Two-frame structure from motion	327
6.2.1	RANSAC (revisited)	328
6.2.2	Self-calibration	328
6.2.3	<i>Application:</i> View morphing	329
6.3	Factorization	329
6.3.1	Projective factorization	330
6.3.2	<i>Application:</i> Sparse 3D model extraction	330
6.4	Bundle adjustment	330
6.4.1	Uncertainty and ambiguities	331
6.4.2	<i>Application:</i> Match move	331
6.5	Constrained structure and motion	331
6.5.1	Line and plane-based techniques	331
6.6	Exercises	332
7	Dense motion estimation	335
7.1	Translational alignment	338
7.1.1	Hierarchical motion estimation	341
7.1.2	Fourier-based alignment	342
7.1.3	Incremental refinement	346
7.2	Parametric motion	351
7.2.1	<i>Application:</i> Video stabilization	356
7.3	Spline-based motion	356
7.3.1	<i>Application:</i> Automated morphing	359
7.4	Optical flow	359
7.4.1	Spatio-temporal filtering	360
7.4.2	<i>Application:</i> Frame interpolation	360
7.5	Layered motion	360
7.6	Learned motion models (tracking, revisited)	362
7.6.1	<i>Application:</i> Motion-based user interface	362
7.7	Exercises	362

8 Image stitching	367
8.1 Motion models	370
8.1.1 Planar perspective motion	370
8.1.2 <i>Application:</i> Whiteboard and document scanning	372
8.1.3 Rotational panoramas	373
8.1.4 <i>Application:</i> Video summarization and compression	377
8.1.5 Cylindrical and spherical coordinates	378
8.2 Global alignment	382
8.2.1 Bundle adjustment	382
8.2.2 Parallax removal	386
8.2.3 Recognizing panoramas	387
8.2.4 <i>Application:</i> Full-view panoramas and virtual environments	388
8.2.5 Direct vs. feature-based alignment	388
8.3 Compositing	392
8.3.1 Choosing a compositing surface	392
8.3.2 Pixel selection and weighting (de-ghosting)	394
8.3.3 Blending	400
8.3.4 <i>Application:</i> Photomontage	403
8.4 Exercises	405
9 Computational photography	407
9.1 Photometric calibration	410
9.1.1 Radiometric response function.	410
9.1.2 Vignetting	414
9.1.3 Optical blur (spatial response estimation)	416
9.2 High dynamic range imaging	419
9.2.1 Tone mapping	427
9.2.2 <i>Application:</i> Flash photography	435
9.3 Super-resolution and blur removal	437
9.3.1 Color image de-mosaicing	438
9.4 Image matting and compositing	439
9.4.1 Blue screen matting	440
9.4.2 Natural image matting	442
9.4.3 Optimization-based matting	446
9.5 Texture analysis and synthesis	451
9.5.1 <i>Application:</i> Hole filling and inpainting	451
9.6 Non-photorealistic rendering	451

9.7 Exercises	452
10 Stereo correspondence	459
10.1 Epipolar geometry	462
10.1.1 Rectification	463
10.1.2 Plane sweep	465
10.2 Sparse correspondence	467
10.3 Dense correspondence	467
10.3.1 Similarity measures	468
10.4 Local methods	469
10.4.1 Sub-pixel estimation and uncertainty	470
10.4.2 <i>Application:</i> Stereo-based head tracking	471
10.5 Global optimization	472
10.5.1 <i>Application:</i> Z-keying (background replacement)	475
10.6 Multi-view stereo	476
10.6.1 Volumetric and 3D surface reconstruction	477
10.6.2 Shape from LightFields (<i>move?</i>)	478
10.7 Exercises	478
11 Shape and appearance modeling	481
11.1 Shape from X	482
11.1.1 Shape from shading and photometric stereo	482
11.1.2 Shape from texture	482
11.1.3 Shape from focus	482
11.2 3D curves and profiles	482
11.3 Volumetric representations	483
11.3.1 Shape from silhouettes	483
11.3.2 Implicit surfaces and level sets (<i>move to multi-view stereo?</i>)	483
11.4 Active rangefinding	484
11.4.1 Range data merging	484
11.5 Surface representations	485
11.5.1 Surface interpolation	485
11.5.2 Surface simplification	486
11.5.3 <i>Application:</i> 3D photography	486
11.6 Point-based representations	486
11.7 Model-based reconstruction	487
11.7.1 Architecture	487

11.7.2 Heads and faces	488
11.7.3 Whole-body modeling and motion (move to Motion?)	488
11.8 Estimating texture maps and albedos (move elsewhere?)	489
11.8.1 Estimating BRDFs	489
11.9 Exercises	489
12 Image-based rendering	491
12.1 View interpolation	492
12.1.1 View-dependent texture maps	492
12.1.2 <i>Application:</i> Photo Tourism	492
12.2 Layered depth images	492
12.2.1 Geometry images	492
12.2.2 Impostors, sprites, and layers	493
12.3 Lightfields and Lumigraphs	494
12.3.1 Unstructured Lumigraph	494
12.3.2 Surface lightfields	494
12.3.3 Concentric and manifold mosaics	495
12.3.4 <i>Application:</i> Synthetic re-focusing	495
12.4 Environment mattes	495
12.5 The modeling / rendering continuum	495
12.6 Exercises	496
13 Video-based rendering	499
13.1 Video enhancement	500
13.1.1 Video denoising	500
13.2 Visual effects	500
13.2.1 Video matting	500
13.2.2 Video morphing	501
13.3 Video-based facial animation	501
13.4 Video textures	501
13.4.1 <i>Application:</i> Animating pictures	501
13.5 3D Video	502
13.5.1 <i>Application:</i> Video-based walkthroughs	502
13.6 Exercises	502
14 Recognition	503
14.1 Face recognition	505

14.1.1 Eigenfaces	506
14.1.2 Active appearance models	506
14.2 Face and object detection	506
14.2.1 <i>Application:</i> Personal photo collections	507
14.3 Instance recognition	507
14.3.1 Geometric alignment	507
14.3.2 Large databases	508
14.3.3 <i>Application:</i> Location recognition	508
14.4 Category recognition	509
14.4.1 Bag of words	509
14.4.2 Part-based models	509
14.4.3 Recognition with segmentation	509
14.4.4 Learning	510
14.4.5 <i>Application:</i> Image search	510
14.5 Context and scene understanding	510
14.6 Recognition databases and test sets	511
14.7 Exercises	511
15 Conclusions (or better title)	513
A Linear algebra and numerical techniques	515
A.1 Matrix algebra	516
A.2 Linear regression	516
A.3 Non-linear regression	516
A.4 Sparse matrix techniques (direct)	516
A.5 Iterative techniques	517
B Estimation theory and Bayesian inference	519
B.1 Estimation theory	521
B.2 Maximum likelihood estimation and least squares	523
B.3 Robust statistics	524
B.4 Prior models and Bayesian inference	524
B.5 Markov Random Fields	525
B.6 Inference algorithms	525
B.6.1 Gradient descent and simulated annealing	525
B.6.2 Dynamic programming	525
B.6.3 Belief propagation	525

B.6.4	Graph cuts	526
B.7	Uncertainty estimation (error analysis)	527
B.8	Kalman filtering	527
B.9	Particle filtering	529
C	Supplementary material	531
C.1	Data sets	532
C.2	Software	532
C.2.1	GPU implementation	534
C.3	Slides	537
C.4	Bibliography	538
	Bibliography	539
	Index	629

Preface

[*Note: This probably belongs after the title and before the Table of Contents, but I'm keeping it here for now so it's easier to get at the TOC.*]

[*Note: Start with this little bit of background, or start with a more direct paragraph describing what this book is about?*]

The seeds for this book were first planted in 2001 when Steve Seitz at the University of Washington invited me to co-teach a course called “Computer Vision for Computer Graphics”. At that time, computer vision techniques were increasingly being used in computer graphics to create image-based models of real-world objects, to create visual effects, and to merge real-world imagery using computational photography techniques. Our decision to focus the applications of computer vision on fun problems such as image stitching and photo-based 3D modeling from your own photos seemed to resonate well with our students.

Since that time, a similar syllabus and project-oriented course structure has been used to teach general computer vision courses both at the University of Washington and at Stanford. (The latter was a course I co-taught with David Fleet in 2003.) Similar curricula have also been adopted at a number of other universities and also incorporated into more specialized courses on computational photography.

This book also reflect my twenty years’ experience doing computer vision research in corporate research labs, mostly at Digital Equipment Corporation’s Cambridge Research Lab and at Microsoft Research. In pursuing my work, I have mostly focused on problems and solution techniques (algorithms) that have practical real-world applications and that work well in practice. Thus, this book has more emphasis on basic techniques that work under real-world conditions, and less on more esoteric mathematics that has intrinsic elegance but less practical applicability.

This book is suitable for teaching a senior-level undergraduate course in computer vision to students in both computer science and electrical engineering. I prefer students that have either an image processing or a computer graphics course as a prerequisite so that they can spend less time learning general background mathematics and more time studying computer vision techniques. The book is also suitable for teaching graduate-level courses in computer vision (by delving into the more demanding application and algorithmic areas), and as a general reference to fundamental

techniques and recent research literature. To this end, I have attempted wherever possible to at least cite the newest research in each sub-field, even if the technical details are too complex to cover in the book itself.

In teaching our courses, we have found it useful for the students to attempt a number of small implementation projects, which often build on one another, in order to get them used to working with real-world images and the challenges that these present. The students are then asked to choose an individual topic for each of their small group final projects. (Sometimes these projects even turn into conference papers!) The exercises at the end of each chapter contain numerous suggestions for both smaller mid-term projects, as well as more open-ended problems whose solution is still an active research area. Wherever possible, I encourage students to try their algorithms on their own personal photographs, since this better motivates them, often leads to creative variants on the problems, and better acquaints them with the variety and complexity of real-world imagery.

In formulating and solving computer vision problems, I have often found it useful to draw inspiration from three different high-level approaches:

1. **Scientific:** build detailed models of the image creation process and develop mathematical techniques to invert these in order to recover the quantities of interest (where necessary, making simplifying assumption to make the mathematics more tractable).
2. **Statistical:** use probabilistic models to quantify the (prior) likelihood of your unknowns and the noisy measurement processes that produce the input images, then infer the best possible estimates of your desired quantities and analyze their resulting uncertainties. The inference algorithms used are often closely related to the optimization techniques used to invert the (scientific) image formation processes.
3. **Engineering:** develop techniques that are simple to describe and implement, but that are also known to work well in practice. Test these techniques to understand their limitation and failure modes, as well as their expected computational costs (run-time performance).

These three approaches build on each other and are used throughout the book.

My personal research and development philosophy (and hence the exercises in the book) have a strong emphasis on *testing* algorithms. It's too easy in computer vision to develop an algorithm that does something *plausible* on a few images rather than something *correct*. The best way to validate your algorithms is to use a three part strategy.

First, test your algorithm on clean synthetic data, for which the exact results are known. Second, add noise to data, and evaluate how the performance degrades as a function of noise level. Finally, test the algorithm on real-world data, preferably drawn from a widely variable source, such as photos found on the Internet. Only then can you truly know if your algorithm can deal with real-world complexity (i.e., images that do not fit some simplified model or assumptions).

In order to help students in this process, this book comes with a large amount of supplementary material, which can be found on the book Web site <http://add-URL-here>. This material, which is described in Appendix C, includes:

- Sample data sets for the problems, including both inputs and desired (or plausible) outputs, as well as all of the images and figures used in this book.
- Pointers to software libraries, which can help students get started with basic tasks such as reading/writing images or creating and manipulating images. Some of these libraries also contain implementations of a wide variety of computer vision algorithms, which can enable you to tackle more ambitious projects (with your instructor's consent, of course :-)).
- Slide sets corresponding to the material covered in this book. [*Note: Until these sets are ready, your best bet is to look at the slides from the courses we have taught at the University of Washington, such as <http://www.cs.washington.edu/education/courses/cse576/08sp/>.*]
- A BibTeX bibliography of the papers cited in this book.

The latter two resources may be of more interest to instructors and researchers publishing new papers in this field, but they will probably come in handy even with regular students.

Acknowledgements

I would like to gratefully acknowledge all of the people whose passion for research and inquiry as well as encouragement have helped me write this book.

Steve Zucker at McGill University first introduced me to computer vision, taught all of his students to question and debate research results and techniques, and encouraged me to pursue a graduate career in this area.

Takeo Kanade and Geoff Hinton, my Ph. D. thesis advisors at Carnegie Mellon University, taught me the fundamentals of good research, writing, presentation. They fired up my interest in visual processing, 3D modeling, and statistical methods, while Larry Matthies introduced me to Kalman filtering and stereo matching.

Demetri Terzopoulos was my mentor at my first industrial research job and taught me the ropes of successful conference publishing. Yvan Leclerc and Pascal Fua, colleagues from my brief interlude at SRI International, gave me new perspectives on alternative approaches to computer vision.

During my six years of research at Digital Equipment Corporation's Cambridge Research Lab, I was fortunate to work with a great set of colleagues, including Ingrid Carlstrom, Gudrun Klinker,

Keith Waters, Richard Weiss, and Stéphane Lavallée, as well as to supervise the first of a long string of outstanding summer interns, including David Tonnessen, Sing Bing Kang, James Coughlan, and Harry Shum. This is also where I began my long-term fruitful collaboration with Daniel Scharstein (now at Middlebury College).

At Microsoft Research, I've had the outstanding fortune to work with some of the world's best researchers in computer vision and computer graphics, including Michael Cohen, Hugues Hoppe, Stephen Gortler, Steve Shafer, Matthew Turk, Harry Shum, Phil Torr, Antonio Criminisi, Ramin Zabih, Shai Avidan, Sing Bing Kang, Matt Uyttendaele, Larry Zitnick, Richard Hartley, Drew Steedly, Matthew Brown, Simon Baker, Dani Lischinski, and Michael Goesele. I was also lucky to have as interns such great students as Polina Golland, Simon Baker, Mei Han, Arno Schödl, Ron Dror, Ashley Eden, Jinxiang Chai, Rahul Swaminathan, Yanghai Tsin, Sam Hasinoff, Anat Levin, Matthew Brown, Vaibhav Vaish, Jan-Michael Frahm, James Diebel, Ce Liu, Josef Sivic, Neel Joshi, Sudipta Sinha, Zeev Farbman, and Rahul Garg.

While working at Microsoft, I've also had the opportunity to collaborate with wonderful colleagues at the University of Washington, where I hold an Affiliate Professor appointment. I'm indebted to David Salesin, who first encouraged me to get involved with the research going on at UW, my long-time collaborators Brian Curless, Steve Seitz, and Maneesh Agrawala, as well as the student's I've had the privilege to supervise and interact with, including Frédéric Pighin, Yung-Yu Chuang, Colin Zheng, Aseem Agarwala, Noah Snavely, Rahul Garg, and Ryan Kaminsky. In particular, as I mentioned at the beginning of this preface, this book owes its inception to the vision course that Steve Seitz invited me to co-teach, as well as to Steve's encouragement, course notes, and editorial input.

I'm also grateful to the many other computer vision researchers who have given me so many constructive suggestions about the book, including Sing Bing Kang, Daniel Scharstein, Bill Freeman, Jana Košecká, Song Yi, and *add your name here*. [Note: See LaTeX source for list of other potential per-chapter reviewers.] If you have any suggestions for improving the book, please send me an e-mail, as I would like to keep the book as accurate, informative, and timely as possible. Keith Price's Annotated Computer Vision Bibliography¹ has proven invaluable in tracking down references and finding related work.

Lastly, this book would not have been possible or worthwhile without the incredible support and encouragement of my family. I dedicate this book to my parents, Zdzisław and Jadwiga, whose love, generosity, and accomplishments have always inspired me; to my sister Basia for her lifelong friendship; and especially to Lyn, Anne, and Stephen, whose daily encouragement in all matters (including this book project) makes it all worthwhile.

¹ <http://iris.usc.edu/Vision-Notes/bibliography/contents.html>

*Lake Wenatchee
July, 2008*

List of symbols

[Note: Insert list of commonly used symbols here (look in *symbols.tex*).]

Chapter 1

Introduction

1.1	A brief history	11
1.2	Overview	19



Figure 1.1: *The human visual system has no problem interpreting the subtle variations in translucency and shading in this photograph and correctly segmenting the object from its background.*

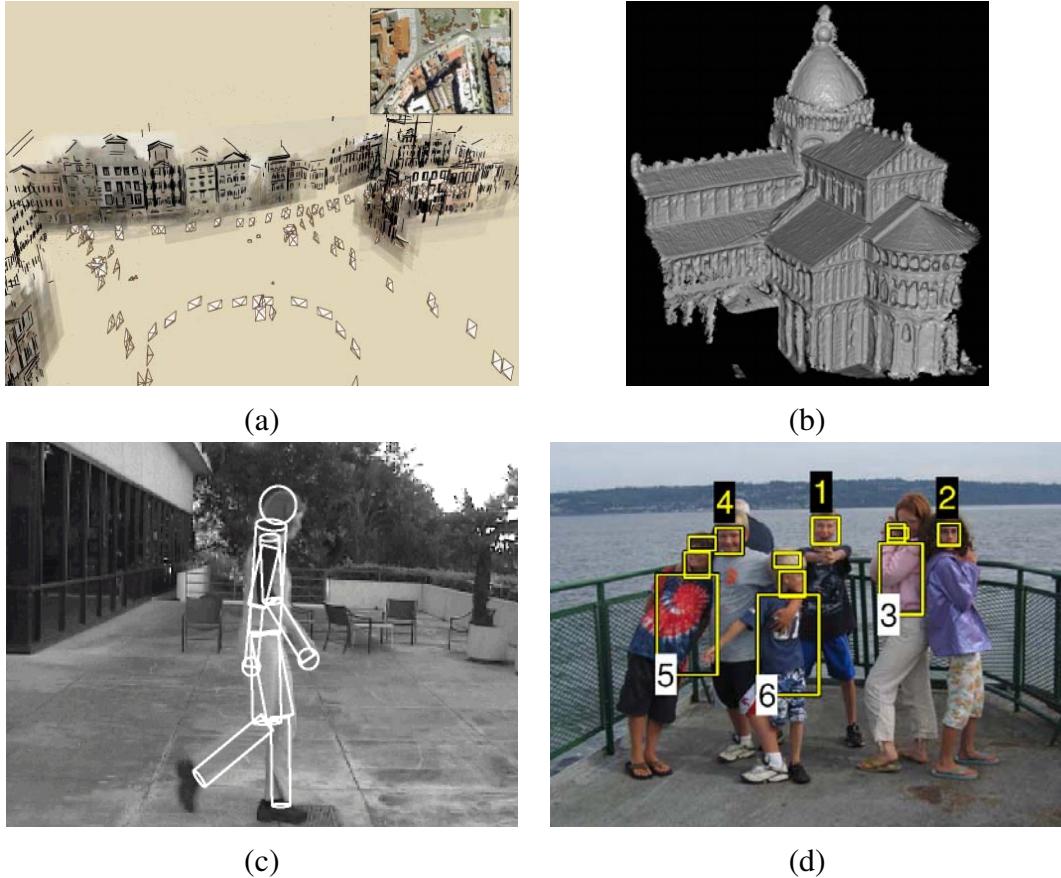


Figure 1.2: *Some examples of computer vision algorithms and applications.* (a) Structure from motion algorithms can reconstruct a sparse 3D point model of a large complex scene from hundreds of partially overlapping photographs (Snavely et al. 2006). (b) Stereo matching algorithms can build a detailed 3D model of a building façade from hundreds of differently exposed photographs taken from the Internet (Goesele et al. 2007). (c) Person tracking algorithms can track a person walking in front of a cluttered background (Sidenbladh and Black 2003). (d) Face detection algorithms, coupled with color-based clothing and hair detection algorithms, can locate and recognize the individuals in this image (Sivic et al. 2006).

As humans, we perceive the three-dimensional structure of the world around us with apparent ease. Think of how vivid the three-dimensional percept is when you look at a vase of flowers sitting on the table next to you. You can tell the shape and translucency of each petal through the subtle patterns of light and shading that play across its surface, and effortlessly segment each flower from the background of the scene (Figure 1.1). Or, looking at a framed group portrait, you can easily count (and name) all of the people in the picture, and even guess at their emotions from their facial appearance. Perceptual psychologists have spent decades trying to understand how the visual system works, and even though they can devise optical illusions to tease apart some of its principles (Figure 1.3), a complete solution to this puzzle remains elusive.

Researchers in computer vision have in parallel been developing mathematical techniques for recovering the three-dimensional shape and appearance of objects in imagery. We now have reliable techniques for accurately computing a partial 3D model of an environment from thousands of partially overlapping photographs (Snavely *et al.* 2006) (Figure 1.2a). Given a large enough set of views of a particular object or façade, we can create accurate dense 3D surface models using stereo matching (Goesele *et al.* 2007) (Figure 1.2b). We can track a person moving against a complex background (Sidenbladh and Black 2003) (Figure 1.2c). We can even, with moderate success, attempt to find and name all of the people in a photograph using a combination of face, clothing, and hair detection and recognition (Sivic *et al.* 2006) (Figure 1.2d). However, despite all of these advances, the dream of having a computer interpret an image at the same level as a two-year old (say counting all of the animals in a picture) remains elusive.

Why is vision so difficult? In part, it's because vision is an *inverse problem*, in which we seek to recover some unknowns given insufficient information to fully specify the solution. We must therefore resort to physics-based and probabilistic *models* to disambiguate between potential solutions. However, modeling the visual world in all of its rich complexity is far more difficult than, say, modeling the vocal tract that produces spoken sounds.

The *forward* models that we use in computer vision are usually developed in physics (radiometry, optics, sensor design) and in computer graphics. Both of these fields model how objects move and animate, how light reflects off their surfaces, is scattered by the atmosphere, refracted through camera lenses (or human eyes), and finally projected onto a flat (or curved) image plane. While computer graphics are not yet perfect (no fully computer-animated movie with human characters has yet succeeded at crossing the *uncanny valley* that separates real humans from android robots and computer animated humans¹), in limited domains, such as rendering a still scene composed of everyday objects, or even animating extinct creatures such as dinosaurs, the illusion of reality *is* perfect.

¹ The term *uncanny valley* was originally coined by roboticist Masahiro Mori as applied to robotics (Mori 1970). It is also commonly applied to computer animated films such as *Final Fantasy* and *Polar Express* (http://en.wikipedia.org/wiki/Uncanny_Valley).

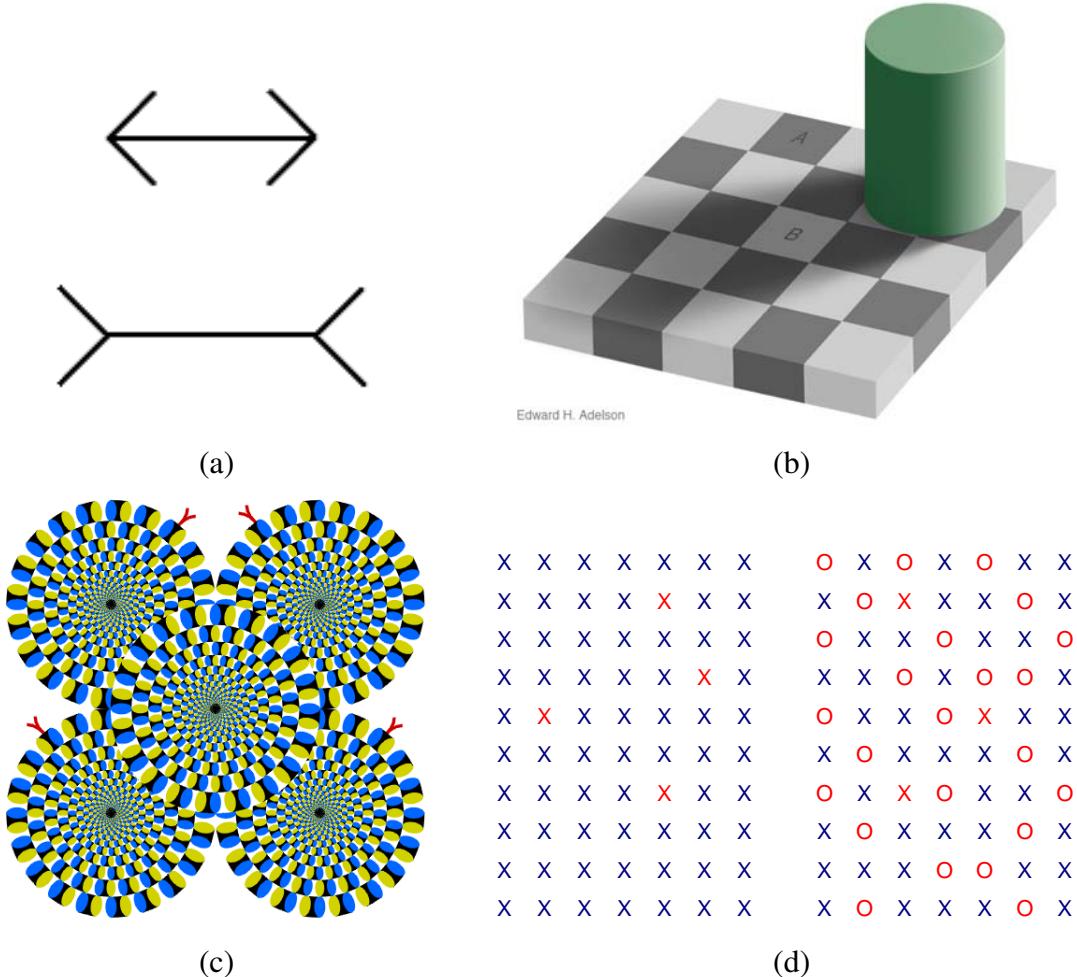


Figure 1.3: Some common optical illusions and what they might tell us about the visual system. (a) The classic Müller-Lyer illusion, where the length of the two horizontal lines appear different, probably due to the imagined perspective effects, http://www.michaelbach.de/ot/sze_muelue. (b) The “white” square B in the shadow and the “black” square A in the light actually have the same absolute intensity value. The percept is due to brightness constancy, the visual system’s attempt to discount illumination when interpreting colors (from http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html). (c) The static image appears to move in a consistent direction, possibly due to differences in transmission speeds between different visual pathways (from <http://www.psy.ritsumei.ac.jp/~akitaoka/rotsnakee.html>). (d) Count the red xs in the left half of the figure. Now count them in the right half. Is it significantly harder? The explanation has to do with a pop-out effect (Treisman 1985), which tells us about the operations of parallel perception and integration pathways in the brain.

[Note: Acknowledge all figures and replace if cannot get permissions.]

In computer vision, we are trying to do the inverse, i.e., to describe the world that we see in one or more images, and to reconstruct its properties such as shape, illumination, and color distributions (Figure 1.10). It is amazing that humans and animals do this so effortlessly, while computer vision algorithms are so error prone. People who have not worked in the field often underestimate the difficulty of the problem. (Colleagues at work often ask me for software to find and name all the people in photos, so they can get on with the more “interesting” work. :-) This misperception that vision should be easy dates back to the early days of artificial intelligence §1.1, when it was initially believed that the *cognitive* (logic proving, planning) parts of intelligence were intrinsically more difficult than the *perceptual* components (Boden 2006).

The good news is that computer vision *is* being used today in a wide variety of real-world applications, which include:

- **Optical character recognition (OCR):** reading handwritten postal codes on letters (Figure 1.4a) and automatic number plate recognition (ANPR)²,
- **Machine inspection:** rapid parts inspection for quality assurance using stereo vision with specialized illumination to measure tolerances on aircraft wings or auto body parts (Figure 1.4b), or looking for defects in steel castings using X-ray vision;
- **Retail:** object recognition for automated checkout lanes (Figure 1.4c),
- **3D model building (photogrammetry):** fully automated 3D model building from aerial photographs used in systems such as Virtual Earth³,
- **Medical imaging:** registering pre-operative and intra-operative imagery (Figure 1.4d), or performing long-term studies of people’s brain morphology as they age;
- **Automotive safety:** detecting unexpected obstacles such as pedestrians on the street, under conditions where active vision techniques such as radar or lidar do not work as well (Figure 1.4e);
- **Match move:** merging computer generated imagery (CGI) with live action footage by tracking feature points in the source video to estimate the 3D camera motion and shape of the environment. Such techniques are widely used in the Hollywood (e.g., in movies such as Jurassic Park) (Roble 1999), and also require the use of precise *matting* to insert new elements between foreground and background elements (Chuang *et al.* 2002).

² http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

³ <http://maps.live.com>

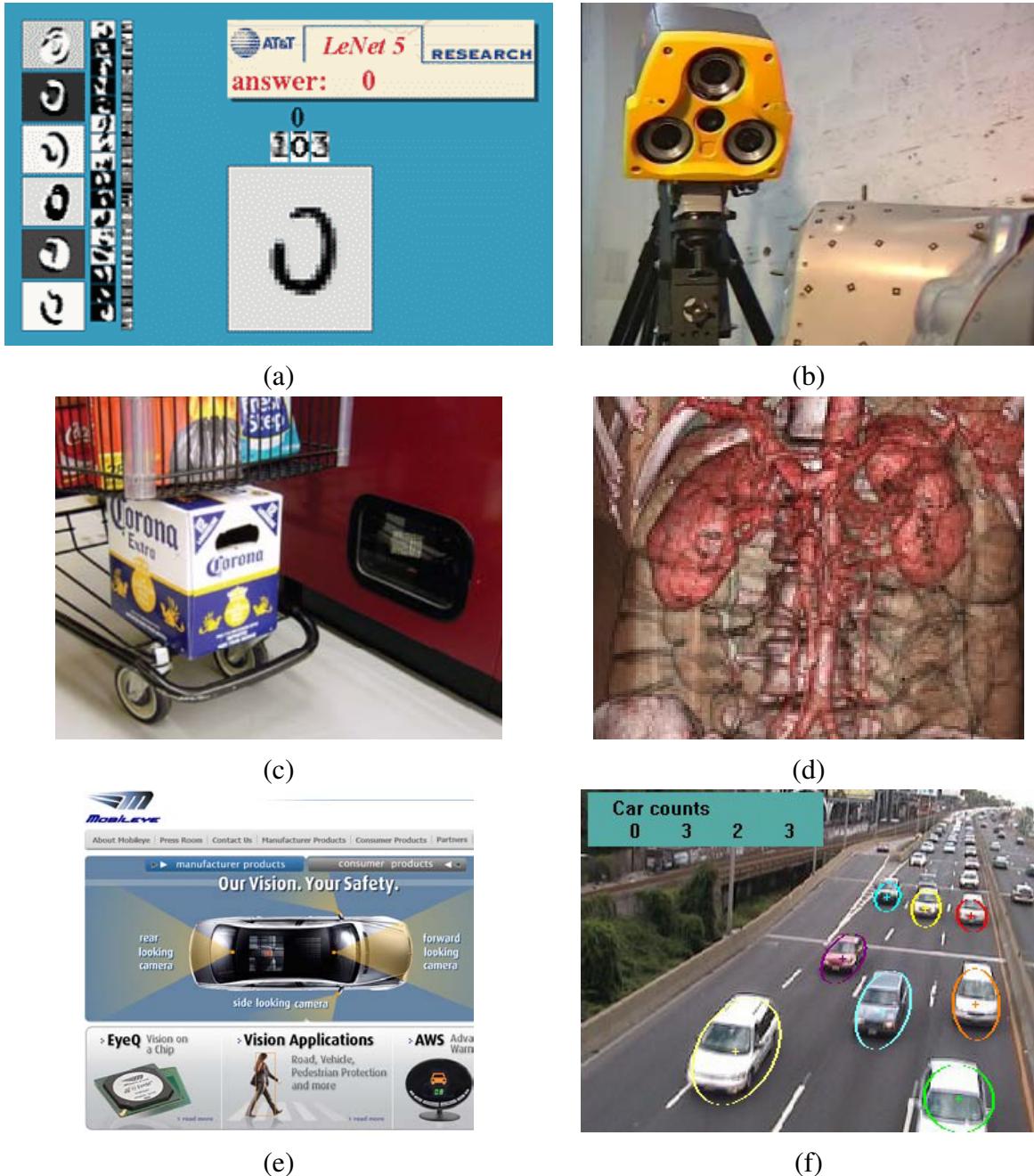


Figure 1.4: Some industrial applications of computer vision <http://www.cs.ubc.ca/spider/lowe/vision.html>: (a) optical character recognition (OCR) <http://yann.lecun.com/exdb/lenet/>; (b) machine inspection <http://www.cognitens.com/>; (c) retail <http://www.evoretail.com/>; (d) medical imaging <http://www.clarontech.com/>; (e) automotive safety <http://www.mobileye.com/>; (f) surveillance and traffic monitoring <http://www.honeywellvideo.com/>.

- **Motion capture (mocap):** of actors for computer animation, using retro-reflective markers viewed from multiple cameras or other vision-based techniques⁴,
- **Surveillance:** monitoring for intruders, analyzing highway traffic (Figure 1.4f), monitoring pools for drowning victims,
- **Fingerprint recognition:** for automatic access authentication as well as forensic applications,
- **Other applications:** David Lowe's web site of industrial vision applications, <http://www.cs.ubc.ca/spider/lowe/vision.html>, lists many other interesting industrial application of computer vision.

While these are all extremely important applications, they mostly pertain to fairly specialized kinds of imagery and narrow domains.

In this book, I focus more on broader *consumer-level* applications, such as fun things you can do with your own personal photographs and video. These include:

- **Stitching:** turning overlapping photos into a single seamlessly stitched panorama, as described in §8 (Figure 1.5a);
- **Exposure bracketing:** merge multiple exposures taken under challenging lighting conditions (strong sunlight and shadows) into a single perfectly exposed image §9.2 (Figure 1.5b);
- **Morphing:** turning one of your friend's pictures into another one's, using a seamless *morph* transition §7.3.1 (Figure 1.5c);
- **3D modeling:** convert one or more snapshots into a 3D model of the object or person you are photographing §11.7 (Figure 1.5d);
- **Video match move and stabilization:** insert 2D pictures or 3D models into your videos by automatically tracking nearby reference points⁵ §6.4.2, or use the motion estimates to remove shake from your videos §7.2.1;
- **Photo-based walkthroughs:** navigate a large collection of photographs, such as the interior of your house, by flying between different photos in 3D (§12.1.2 and §13.5.1)
- **Face detection:** for improved camera focusing as well as more relevant image search §14.2;

⁴ http://en.wikipedia.org/wiki/Motion_capture

⁵ For a fun student project on this topic, see the "PhotoBook" project at <http://www.cc.gatech.edu/dvfx/videos/dvfx2005.html>.

- **Visual authentication:** automatically log different family members onto your home computer as they sit down in front of the Web cam §14.1.

The great thing about these applications is that they are already familiar to most students, or at least they are technologies that they can immediately appreciate and use with their own personal media. Since computer vision is a challenging topic, given the wide range of mathematics being covered⁶ and the intrinsically difficult nature of the problems being solved, having fun and relevant problems to work on can be highly motivating and inspiring.

The other major reason why this book has a strong focus on applications is that these can be used to *formulate* and *constrain* the potentially open-ended problems endemic in vision. For example, if someone comes to me and asks for a good edge detector, my first question is usually to ask *why*? What kind of problem are they trying to solve, and why do they believe that edge detection is an important component?

If they are trying to locate faces §14.2, I usually point out that most successful face detectors use a combination of skin color detection (Exercise 2.9) and simple blob features (§14.2), and do not rely on edge detection.

If they are trying to match door and window edges in a building for the purpose of 3D reconstruction, I tell them that edges are a fine idea, but it's better to tune the edge detector for long edges §3.2.1, §4.2, and then link these together into straight lines with common vanishing points before matching §4.3.

Thus, it's better to think back from problem at hand to suitable techniques, rather than to grab the first technique that you may have heard of. This kind of working back from problems to solutions is typical of an *engineering* approach to the study of vision, and reflects my own background in the field. First, I come up with a detailed problem definition and decide on the constraints and/or specifications for the problem. Then, I try to find out which techniques are known to work, implement a few of these, and finally evaluate their performance and make a selection. In order for this process to work, it's important to have realistic test data, both synthetic, which can be used to verify correctness and analyze noise sensitivity, and real-world data typical of the way the system will finally get used.

However, this book is not just an engineering text (a source of recipes). It also takes a *scientific* approach to the basic vision problems. Here, I try to come up with the best possible models of the physics of the system at hand: how the scene is created, how light interacts with the scene and atmospheric effects, and how the sensors work, including sources of noise and uncertainty. The task is then to try to invert the acquisition process to come up with the best possible description of the scene.

⁶ These techniques include physics, Euclidean and projective geometry, statistics, and optimization, and make computer vision a fascinating field to study and a great way to learn techniques widely applicable in other fields.

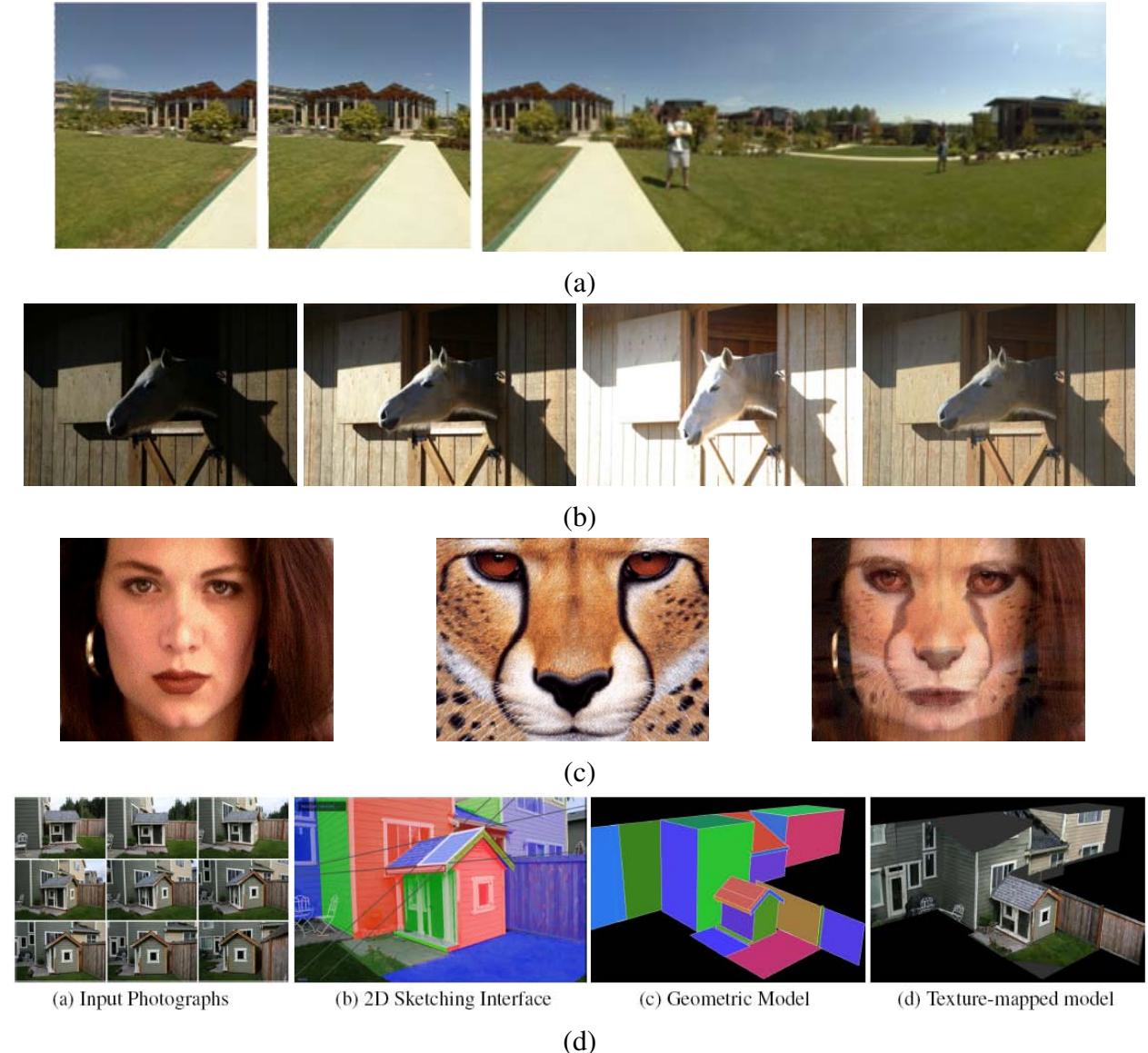


Figure 1.5: *Some consumer applications of computer vision: (a) image stitching: merging different views; (b) exposure bracketing: merging different exposures; (c) morphing: blending between two photographs; (d) turning a collection of photographs into a 3D model.*

The book often uses a *statistical* approach to formulating and solving computer vision problems. Where appropriate, probability distributions are used to model the scene and the noisy image acquisition process. The association of prior distributions with unknowns is often called *Bayesian modeling* (Appendix B). It is even possible to associate a risk or loss function with mis-estimating the answer §B.2, and setting up your inference algorithm to minimize the expected risk. (Consider a robot trying to estimate the distance to an obstacle: it's usually safer to underestimate than to overestimate.) With statistical techniques, it often helps to gather lots of training data from which to learn probabilistic models. Finally, statistical approaches enable you to use proven inference techniques to estimate the best answer (or distribution of answers), and also to quantify the uncertainty in the resulting estimates.

Because so much of computer vision involves the solution of inverse problems or the estimation of unknown quantities, my book also has a heavy emphasis on *algorithms*, especially those that are known to work well in practice. For many vision problems, it is all too easy to come up with a mathematical description of the problem that either does not match realistic real-world conditions or does not lend itself to the stable estimation of the unknowns. What we need are both algorithms that are *robust* to noise and deviation from our models, as well as reasonably *efficient* in terms of run-time and space. In this book, I go into these issues in detail, using Bayesian techniques, where applicable, to ensure robustness, and efficient search, minimization, and linear system solving algorithms to ensure efficiency.

Now that I've described the goals of this book and the frameworks that I use, let me devote the rest of this chapter to two additional topics. The first is a brief synopsis of the history of computer vision, §1.1. This can easily be skipped by those who want to get "to the meat" of the new material in this book and do not care as much about who invented what when.

The second is an overview of the book's contents, §1.2, which is useful reading for everyone who intends to make a study of this topic (or to jump in partway, since it describes inter-chapter dependencies). This outline is also useful for instructors looking to structure one or more courses around this topic, as it provides sample curricula based on the book's contents.

To support the book's use as a textbook, the appendices and associated web site contain lots of additional course materials, including slide sets, test images and solutions, and pointers to software. The book chapters have exercises, some suitable as written homework assignments, others as shorter one-week projects, and still others as open-ended research problems suitable as challenging final projects.

As a reference book, I try wherever possible to discuss which techniques and algorithms work well in practice, as well as provide up-to-date pointers to the latest research results in the areas that I cover. The exercises can also be used to build up your own personal library of self-tested and validated vision algorithms, which in the long term (assuming you have the time), is more worthwhile than simply pulling algorithms out of a library whose performance you do not really

understand.

1.1 A brief history

[Note: This whole section could definitely use some sanity checking from other researchers who know the older literature better. It's also pretty heavy on references, so I could move some of these to a bibliography section.]

In this section, I provide a brief personal synopsis of the main development in computer vision over the last thirty years, or at least those that I find personally interesting and which appear to have stood the test of time. Readers not interested as much in the provenance of various ideas and the evolution of this field should skip ahead to the book overview (§1.2).

1970s. When computer vision first started out in the early 1970s, it was viewed as the visual perception component of an ambitious agenda to mimic human intelligence and to endow robots with intelligent behavior. At the time, it was believed by some of the early pioneers of artificial intelligence and robotics (at places like MIT, Stanford, and CMU) that solving the “visual input” problem would be an easy step along the path to solving more difficult problems such as higher-level reasoning and planning. (According to one well-known story, in 1966, Marvin Minsky at MIT asked his undergraduate student Gerald Jay Sussman to “spend the summer linking a camera to a computer and getting the computer to describe what it saw” (Boden 2006, p. 781).⁷ We now know that the problem is slightly more difficult than that.)

What distinguished computer vision from the already existing field of digital image processing (Rosenfeld and Kak 1976) was a desire to recover the three-dimensional structure of the world from images, and to use this as a stepping stone towards full scene understanding. Winston (1975) and Hanson and Riseman (1978) provide two nice collections of classic papers from this early period.

Early attempt at scene understanding involved extracting edges and then inferring the 3D structure of an object from the topological structure of the 2D lines (Roberts 1965). Several *line labeling* algorithms were developed at that time (Huffman 1971, Clowes 1971, Waltz 1975, Rosenfeld *et al.* 1976, Kanade 1980) (Figure 1.6a). Nalwa (1993) gives a nice review of this area. The topic of edge detection (§4.2) was also an active area of research; a nice survey on contemporaneous work can be found in (Davis 1975).

Three-dimensional modeling of non-polyhedral objects was also being studied (Baumgart 1974, Baker 1977). One popular approach used *generalized cylinders*, i.e., solid of revolutions and swept

⁷ Boden (2006) cites (Crevier 1993) as the original source.

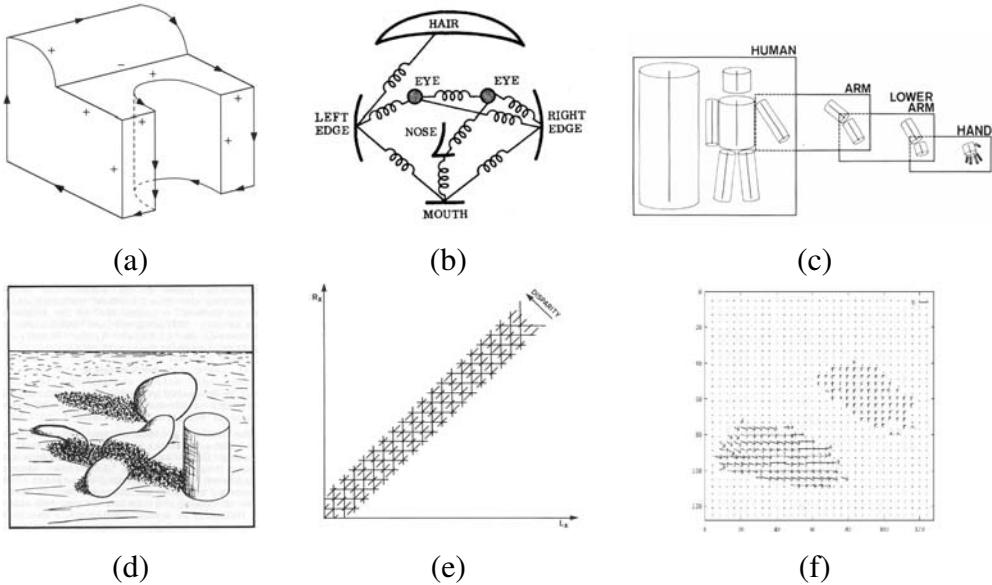


Figure 1.6: Some early examples of computer vision algorithms (1970s): (a) line labeling (Nalwa 1993), (b) pictorial structures (Fischler and Elschlager 1973), (c) articulated body model (Marr 1982), (d) intrinsic images (Barrow and Tenenbaum 1981), (e) stereo correspondence (Marr 1982), (f) optical flow (Nagel and Enkelmann 1986).

closed curves (Agin and Binford 1976, Nevatia and Binford 1977) often arranged into parts relationship⁸ (Hinton 1977, Marr 1982) (Figure 1.6c). Fischler and Elschlager (1973) called such *elastic* arrangements of parts *pictorial structures* (Figure 1.6b). These are currently one of the favored approaches being used in object recognition §14.4 (Felzenszwalb and Huttenlocher 2005).

More quantitative approaches to computer vision were also developed at the time, including the first of many feature-based stereo correspondence algorithms §10.2 (Dev 1974, Marr and Poggio 1976, Moravec 1977, Marr and Poggio 1979, Mayhew and Frisby 1981, Baker 1982, Barnard and Fischler 1982, Ohta and Kanade 1985, Grimson 1985, Pollard *et al.* 1985, Prazdny 1985) (Figure 1.6e), as well as intensity-based optical flow algorithms §7.4 (Horn and Schunck 1981, Huang 1981, Lucas and Kanade 1981, Nagel 1986) (Figure 1.6f). The early work in simultaneously recovering 3D structure and camera motion §6 also began around this time (Ullman 1979, Longuet-Higgins 1981).

A more qualitative approach to understanding intensities and shading variations, and explaining these by the effects of image formation phenomena such as surface orientation and shadows was championed by (Barrow and Tenenbaum 1981) in their paper on *intrinsic images* (Figure 1.6d), along with Marr's (1982) related *2½-D sketch* ideas. This approach is again seeing a bit of a revival in the work of Tappen *et al.* (2005).

⁸ In robotics and computer animation, these linked part graphs are often called *kinematic chains*.

A lot of the philosophy of how vision was believed to work at the time is summarized in David Marr's (1982) book *Vision*. In particular, Marr introduced his notion of the three levels of description of a (visual) information processing system. These three levels, very loosely paraphrased according to my own interpretation, are:

- **Computational theory:** What is the goal of the computation (task) and what are the constraints that are known or can be brought to bear on the problem?
- **Representations and algorithms:** How are the input, output, and intermediate information represented, and which algorithms are used to calculate the desired result?
- **Hardware implementation:** How are the representations and algorithms mapped onto actual hardware, e.g., a biological vision system or a specialized piece of silicon? Conversely, how can hardware constraints be used to guide the choice of representation and algorithm? With the increasing use of graphics chips (GPUs) and many-core architectures for computer vision §C.2.1, this question is again becoming quite relevant.

As I mentioned earlier in this introduction, it is my conviction that a careful analysis of the problem specification and known constraints from image formation and priors (the scientific and statistical approaches) must be married with efficient and robust algorithms (the engineering approach) to design successful vision algorithms. Thus, it seems that Marr's philosophy is as good a guide to framing and solving problems in our field today as it was twenty-five years ago.

1980s. In the 1980s, a lot more attention was focused on more sophisticated mathematical techniques for performing quantitative image and scene analysis.

Image pyramids §3.4 started being widely used to perform tasks such as image blending and coarse-to-fine correspondence search (Rosenfeld 1980, Burt and Adelson 1983a, Burt and Adelson 1983b, Rosenfeld 1984, Quam 1984, Anandan 1989) (Figure 1.7a). Continuous version of pyramid using the concept of *scale-space* processing were also developed (Witkin 1983, Witkin *et al.* 1986, Lindeberg 1990). In the late 80s, wavelets §3.4.3 started displacing or augmenting regular image pyramids in some applications (Adelson *et al.* 1987, Mallat 1989, Simoncelli and Adelson 1990a, Simoncelli and Adelson 1990b, Simoncelli *et al.* 1992).

The use of stereo as a quantitative shape cue was extended by a wide variety of *shape-from-X* techniques, including shape from shading §11.1.1 (Horn 1975, Pentland 1984, Blake *et al.* 1985, Horn and Brooks 1986, Horn and Brooks 1989) (Figure 1.7b), photometric stereo §11.1.1 (Woodham 1981) shape from texture §11.1.2 (Witkin 1981, Pentland 1984, Malik and Rosenholtz 1997), and shape from focus §11.1.3 (Nayar *et al.* 1995). Horn (1986) has a nice discussion of most of these techniques.

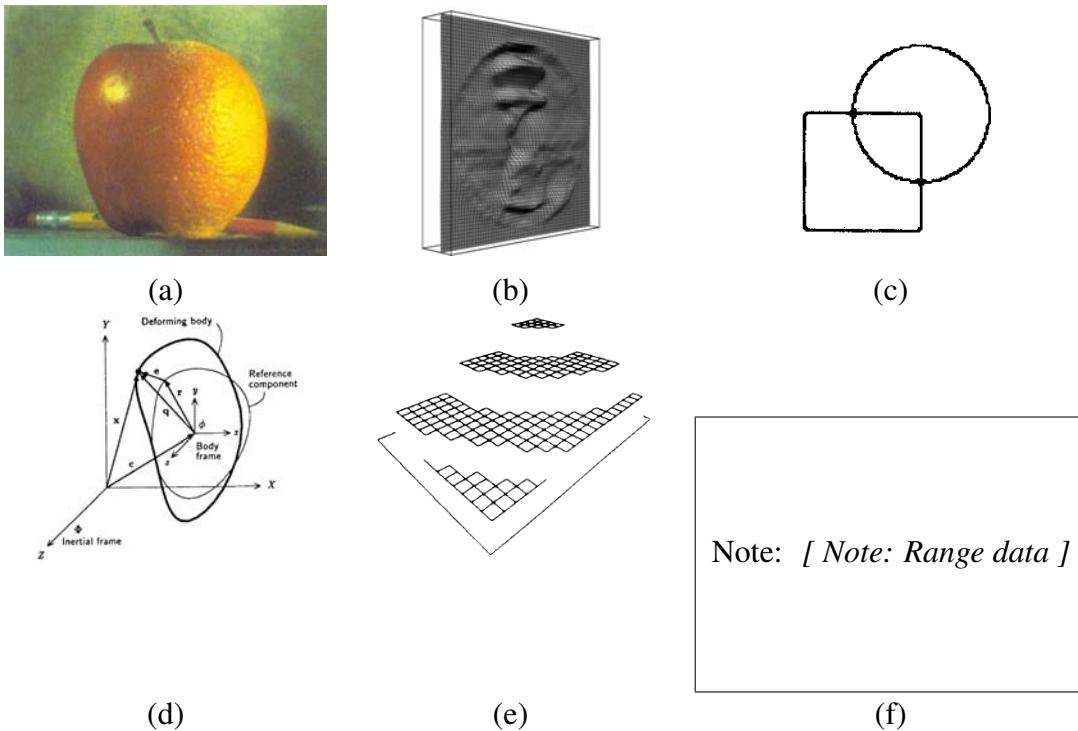


Figure 1.7: Examples of computer vision algorithms from the 1980s: (a) pyramid blending (Burt and Adelson 1983b), (b) shape from shading (Freeman and Adelson 1991), (c) edge detection (Freeman and Adelson 1991), (d) physically-based models (Terzopoulos and Witkin 1988), (e) regularization-based surface reconstruction (Terzopoulos 1988), (f) range data acquisition and merging.

Research into better edge and contour detection §4.2 was also active during this period, (Canny 1986, Nalwa and Binford 1986) (Figure 1.7c), including the introduction of dynamically evolving contour trackers §4.4.1 such as *snakes* (Kass *et al.* 1988), as well as three-dimensional *physically-based models* (Terzopoulos *et al.* 1987, Kass *et al.* 1988, Terzopoulos and Fleischer 1988, Terzopoulos *et al.* 1988) (Figure 1.7d).

Researchers noticed that a lot of the stereo, flow, shape-from-X, and edge detection algorithms could be unified, or at least described, using the same mathematical framework, if they were posed as variational optimization problems §3.6, and made more robust (well-posed) using regularization §3.6.1 (Terzopoulos 1983, Poggio *et al.* 1985, Terzopoulos 1986, Blake and Zisserman 1987, Terzopoulos 1988). Around the same time, Geman and Geman (1984) pointed out that such problems could equally well be formulated using discrete *Markov Random Field* (MRF) models §3.6.2, which enabled the use of better (global) search and optimization algorithms such as simulated annealing.

Online variants of MRF algorithms that modeled and updated uncertainties using the Kalman

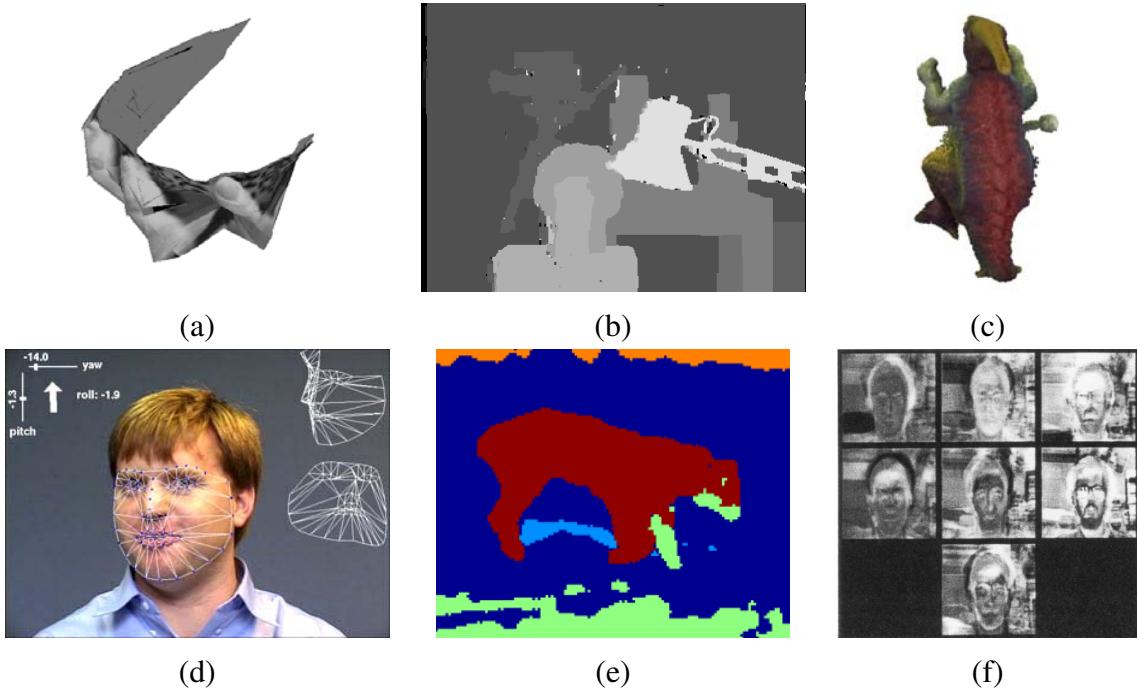


Figure 1.8: Examples of computer vision algorithms from the 1990s: (a) factorization-based structure from motion (Tomasi and Kanade 1992), (b) dense stereo matching (Boykov et al. 2001), (c) multi-view reconstruction (Seitz and Dyer 1999), (d) face tracking (Matthews and Baker 2004, Matthews et al. 2007), (e) image segmentation (Fowlkes et al. 2004), (f) face recognition (Turk and Pentland 1991a).

filter were introduced a little later (Dickmanns and Graefe 1988, Matthies *et al.* 1989, Szeliski 1989). Attempts were also made to map both regularized and MRF algorithms onto parallel hardware (Poggio and Koch 1985, Poggio *et al.* 1988, Fischler *et al.* 1989). The book by Fischler and Firschein (1987) contains a very nice collection of articles focusing on all of these topics (stereo, flow, regularization, MRFs, and even higher-level vision).

Three-dimensional range data processing (acquisition, merging, modeling, and recognition) continued being actively explored during this decade (Agin and Binford 1976, Besl and Jain 1985, Faugeras and Hebert 1987, Curless and Levoy 1996) (Figure 1.7f). The compilation by Kanade (1987) contains a lot of the interesting papers in this area.

1990s. While a lot of the previously mentioned topics continued being explored, a few of them became significantly more active.

A burst of activity in using projective invariants for recognition (Mundy and Zisserman 1992) evolved into a concerted effort to solve the structure from motion problem §6. A lot of the initial activity was directed at *projective* reconstructions, which did not require the knowledge of cam-

era calibration (Faugeras 1992, Hartley *et al.* 1992, Hartley 1994b, Faugeras and Luong 2001, Hartley and Zisserman 2004). Simultaneously, *factorization* techniques §6.3 were developed to efficiently solve problems for which orthographic camera approximations were applicable (Tomasi and Kanade 1992, Poelman and Kanade 1997, Anandan and Irani 2002) (Figure 1.8a) and then later extended to the perspective case (Christy and Horaud 1996, Triggs 1996). Eventually, the field started using full global optimization §6.4 (Taylor *et al.* 1991, Szeliski and Kang 1994, Azarbayejani and Pentland 1995), which was later recognized as being the same as the *bundle adjustment* techniques traditionally used in photogrammetry (Triggs *et al.* 1999). Fully automated (sparse) 3D modeling systems were built using such techniques (Beardsley *et al.* 1996, Schaffalitzky and Zisserman 2002, Brown and Lowe 2003b, Snavely *et al.* 2006).

Work begun in the 1980s on using detailed measurements of color and intensity combined with accurate physical models of radiance transport and color image formation created its own subfield known as *physics-based vision*. A good survey of the field can be found in the three volume collection on this topic (Wolff *et al.* 1992a, Healey and Shafer 1992, Shafer *et al.* 1992).

Optical flow methods §7 continued being improved (Nagel and Enkelmann 1986, Bolles *et al.* 1987, Horn and Weldon Jr. 1988, Anandan 1989, Bergen *et al.* 1992, Black and Anandan 1996, Bruhn *et al.* 2005, Papenberg *et al.* 2006), with (Nagel 1986, Barron *et al.* 1994, Baker *et al.* 2007) being good surveys. Similarly, a lot of progress was made on dense stereo correspondence algorithms §10 (Okutomi and Kanade 1993, Okutomi and Kanade 1994, Boykov *et al.* 1998, Birchfield and Tomasi 1999a, Boykov *et al.* 2001) (see (Scharstein and Szeliski 2002) for a survey and comparison), with the biggest breakthrough being perhaps global optimization using *graph-cut* techniques (Boykov *et al.* 2001) (Figure 1.8b).

Multi-view stereo algorithms that produce complete 3D surfaces §10.6 were also an active topic of research (Seitz and Dyer 1999, Kutulakos and Seitz 2000) (Figure 1.8c) that continues being active today (Seitz *et al.* 2006). Techniques for producing 3D volumetric descriptions from binary silhouettes §11.3.1 continued being developed (Potmesil 1987, Srivasan *et al.* 1990, Szeliski 1993, Laurentini 1994), along with techniques based on tracking and reconstructing smooth occluding contours §11.2 (Cipolla and Blake 1992, Vaillant and Faugeras 1992, Zheng 1994, Boyer and Berger 1997, Szeliski and Weiss 1998, Cipolla and Giblin 2000).

Tracking algorithms also improved a lot, including contour tracking using *active contours* §4.4 such as *snakes* (Kass *et al.* 1988), *particle filters* (Blake and Isard 1998), and *level sets* (Malladi *et al.* 1995), as well as intensity-based (*direct*) techniques (Lucas and Kanade 1981, Shi and Tomasi 1994, Rehg and Kanade 1994), often applied to tracking faces (Lanitis *et al.* 1997, Matthews and Baker 2004, Matthews *et al.* 2007) (Figure 1.8d) and whole bodies (Sidenbladh and Black 2003) [*Note: better references?*] (Figure 1.2c).

Image segmentation §4.5 (Figure 1.8e), a topic which has been active since the earliest days of computer vision (Brice and Fennema 1970, Riseman and Arbib 1977, Rosenfeld and Davis

1979, Haralick and Shapiro 1985), was also an active topic of research, including techniques based on minimum energy (Mumford and Shah 1989) and minimum description length (Leclerc 1989), *normalized cuts* (Shi and Malik 2000), and *mean shift* (Comaniciu and Meer 2002).

Statistical learning techniques started appearing, first in the application of principal component *eigenface* analysis applied to face recognition §14.1.1 (Turk and Pentland 1991a) (Figure 1.8f) and linear dynamical systems for curve tracking §4.4.1 (Blake and Isard 1998).

Perhaps the most notable development in computer vision during this decade was the increased interaction with computer graphics (Seitz and Szeliski 1999), especially in the cross-disciplinary area of *image-based modeling and rendering* §12. The idea of manipulating real-world imagery directly to create new animations first came to prominence with *image morphing* techniques §3.5.3 (Beier and Neely 1992) (Figure 1.5c) and was later applied to *view interpolation* (Chen and Williams 1993, Seitz and Dyer 1996), panoramic image stitching §8 (Mann and Picard 1994, Chen 1995, Szeliski 1996, Szeliski and Shum 1997, Szeliski 2006a) (Figure 1.5a), and full light-field rendering §12.3 (Gortler *et al.* 1996, Levoy and Hanrahan 1996, Shade *et al.* 1998) (Figure 1.9a). At the same time, image-based modeling techniques for automatically creating realistic 3D models from collections of images were also being introduced (Beardsley *et al.* 1996, Debevec *et al.* 1996, Taylor *et al.* 1996) (Figure 1.9b).

2000s. This past decade has continued to see a deepening interplay between the vision and graphics fields. In particular, many of the topics introduced under the rubric of image-based rendering, such as image stitching §8, light-field capture rendering §12.3, and *high dynamic range* (HDR) image capture through exposure bracketing §9.2 (Mann and Picard 1995, Debevec and Malik 1997) (Figure 1.5b), were re-christened as *computational photography* §9 to acknowledge the increased use of such techniques in everyday digital photography. For example, the rapid adoption of exposure bracketing to create high dynamic range images necessitated the development of *tone mapping* algorithms §9.2.1 to convert such images back to displayable results (Fattal *et al.* 2002, Durand and Dorsey 2002, Reinhard *et al.* 2002, Lischinski *et al.* 2006a) (Figure 1.9c). In addition to merging multiple exposures, techniques were developed to merge flash images with non-flash counterparts (Eisemann and Durand 2004, Petschnigg *et al.* 2004), and to interactively or automatically select different regions from overlapping images (Agarwala *et al.* 2004).

Texture synthesis §9.5 and quilting (Efros and Leung 1999, Efros and Freeman 2001, Kwatra *et al.* 2003) (Figure 1.9d) as well as in-painting (Bertalmio *et al.* 2000, Bertalmio *et al.* 2003, Criminisi *et al.* 2004) are two additional topics that can be classified as computational photography techniques, since they re-combine input image samples to produce novel photographs.

A second notable trend during this past decade was the emergence of feature-based techniques (combined with learning) for object recognition §14.3 (Ponce *et al.* 2007). Some of the notable papers in this area include the *constellation model* of Fergus *et al.* (2003), Fergus *et al.* (2005),

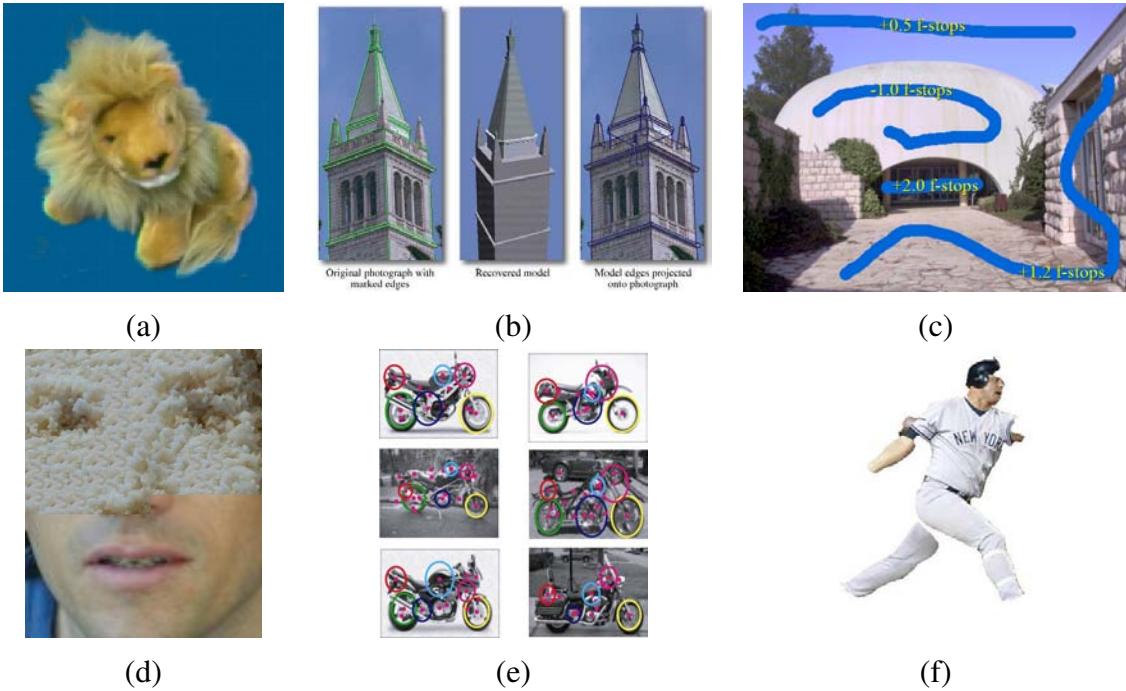


Figure 1.9: *Recent examples of computer vision algorithms:* (a) image-based rendering ([Gortler et al. 1996](#)), (b) image-based modeling ([Debevec et al. 1996](#)), (c) interactive tone mapping ([Lischinski et al. 2006a](#)), (d) texture synthesis ([Efros and Freeman 2001](#)), (e) feature-based recognition ([Fergus et al. 2003](#)), (f) region-based recognition ([Mori et al. 2004](#)).

[Fei-Fei et al. \(2006\)](#) (Figure 1.9e) and the *pictorial structures* of [Felzenszwalb and Huttenlocher \(2005\)](#). Feature-based techniques also dominate other recognition tasks such as scene recognition ([Zhang et al. 2007](#)) and panorama and location recognition ([Brown and Lowe 2007](#), [Schindler et al. 2007](#)). And while *interest point* (patch-based) features tend to dominate current research, some groups are pursuing recognition based on contours ([Belongie et al. 2002](#)) and region segmentation ([Mori et al. 2004](#)) (Figure 1.9f).

Another significant trend from this past decade has been the development of more efficient algorithms for complex global optimization problems §3.6, §B.6, ([Szeliski et al. 2008c](#)). While this trend began with work on graph cuts ([Boykov et al. 2001](#), [Kohli and Torr 2005](#)), a lot of progress has also been made in message passing algorithms such as *loopy belief propagation* (LBP) ([Yedidia et al. 2000](#), [Kumar and Torr 2006](#)).

1.2 Overview

In this final part of this introduction, I give a brief tour of the material in this book, as well as a few notes on notation and some additional general references. Since computer vision is such a broad field, it is possible to study certain aspects of it, e.g., geometric image formation and 3D structure recovery, without engaging other parts, e.g., the modeling of reflectance and shading. Some of the chapters in this book are only loosely coupled with others, and it is not strictly necessary to read all of the material in sequence.

Figure 1.10 shows a rough layout of the contents of this book. Since computer vision involves going from images to a structural description of the scene (and computer graphics the converse), I have positioned the chapter horizontally in terms of which major component they address, in addition to vertically, according to their dependence.

Going from left to right, we see the major column headings as Images (which are 2D in nature), Geometry (which encompasses 3D descriptions), and Photometry (which encompasses object appearance). (An alternative labeling for these latter two could also be *shape* and *appearance*—see, e.g., §12 and (Kang *et al.* 2000).) Of course, this taxonomy should be taken with a large grain of salt, and does not encompass more semantic tasks such as recognition. The placement of topics along the horizontal axis should also be taken lightly, as most vision algorithms involve mapping between at least two different representations. [*Note: One could argue that photometry is actually closer to images than geometry, but since we usually go to geometry first, this seems easier to parse.*]]

Interspersed throughout the book are sample *Applications*, which relate the algorithms and mathematical material being presented in various chapters to useful, real-world applications. Many of these applications are also presented in the exercises sections, so that students can write their own.

At the end of each section, I provide a set of *Exercises* that the students can use to implement, test, and refine the algorithms and techniques presented in each section. Motivated students who implement a reasonable subset of these exercises will by the end of the book have a computer vision software library that can be used for a variety of interesting tasks and projects.

The book begins in Chapter 2 with a review of the image formation processes that create the images that we see and capture. Understanding this process is fundamental if you want to take a scientific (model-based) approach to computer vision. Students who are eager to just start implementing algorithms (or courses that have limited time) can skip ahead to the next chapter and dip into this material later.

In Chapter 2, I break down image formation into three major components. Geometric image formation §2.1 deals with points, lines, and planes, and how these are mapped onto images using *projective geometry* and other models (including radial lens distortion). Photometric image for-

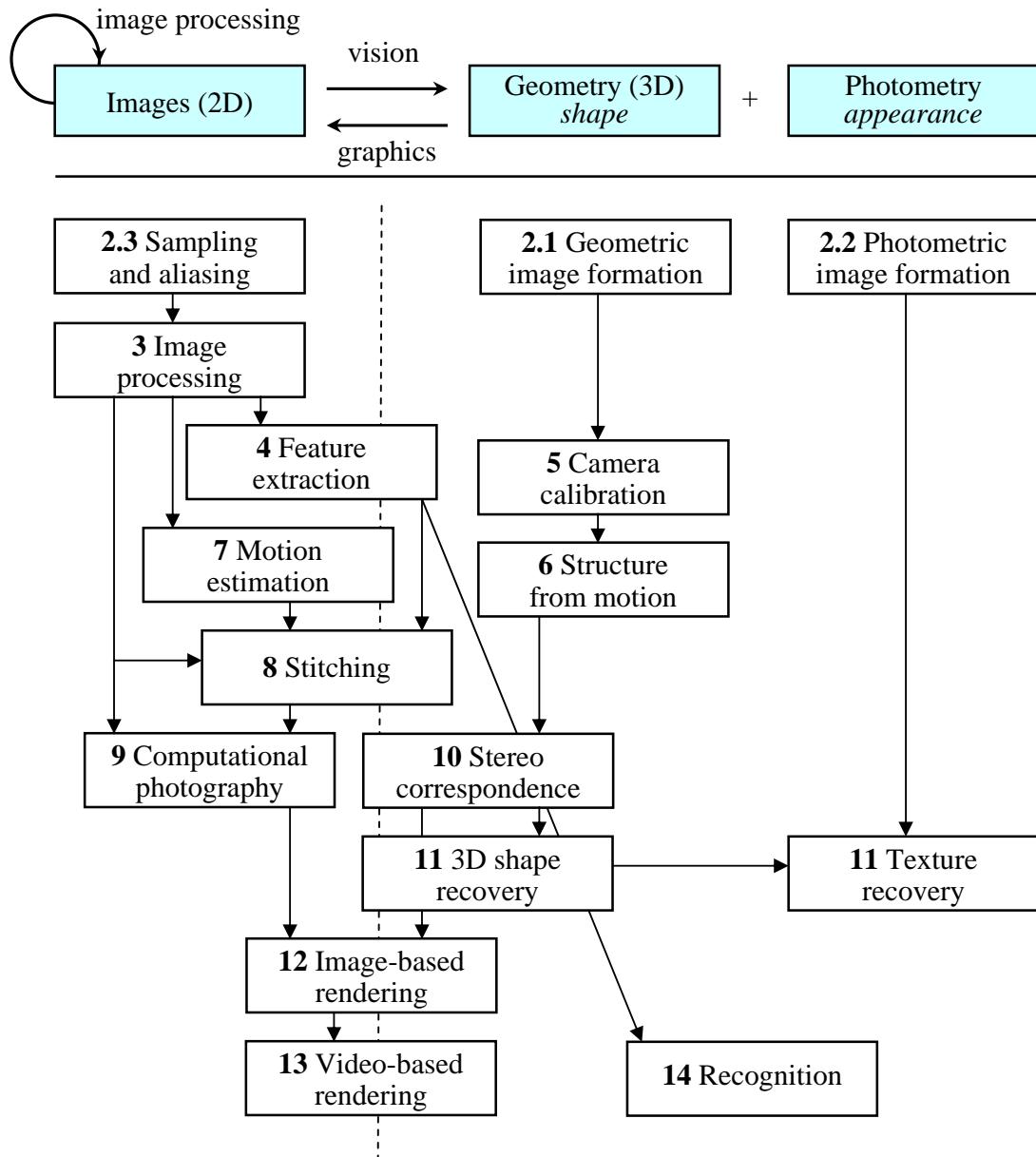


Figure 1.10: Relationship between images, geometry, and photometry, as well as a taxonomy of the topics covered in this book. Topics are roughly positioned along the left-right axis depending on whether they are more closely related to image-based (left), geometry-based (middle) or appearance-based (right) representations.

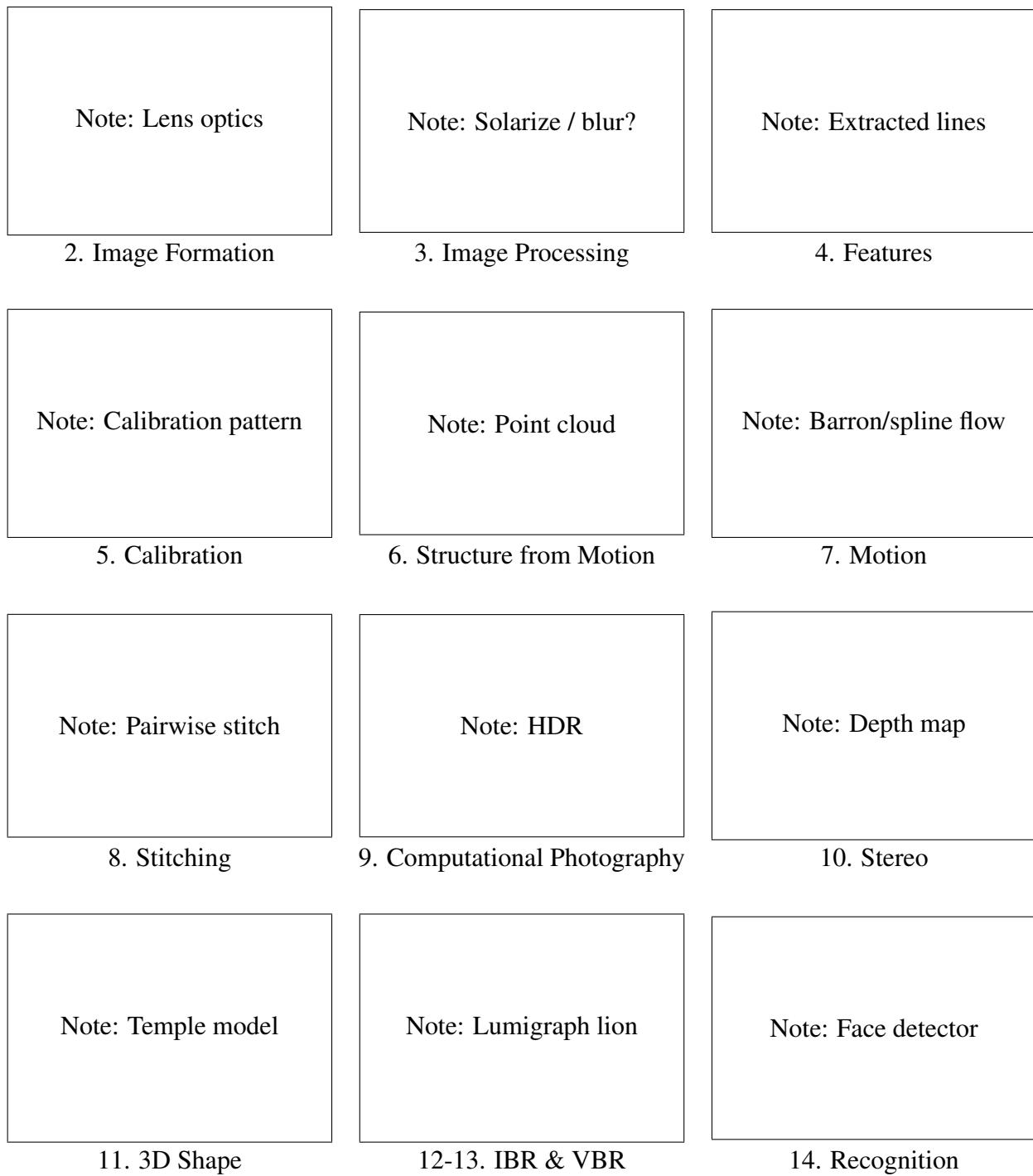


Figure 1.11: A pictorial summary of the chapter contents.

[Note: Move this to earlier in the book, say after overview page? Or, incorporate into (an expanded) overview page, by associating a picture with each chapter, and listing section headings under each chapter name [Nice's Pen-Ink-Watercolor book]?]

mation §2.2 covers *radiometry*, which describes how light interacts with surfaces in the world, as well as *optics*, which projects light onto the sensor plane. Finally, §2.3 covers how sensors work, including topics such as sampling and aliasing, color sensing, and in-camera compression.

Chapter 3 covers image processing, which is needed in almost all computer vision applications. This includes topics such as linear and non-linear filtering §3.2, the Fourier transform §3.3, image pyramids and wavelets §3.4, geometric transformations such as image warping §3.5, and global optimization techniques such as *regularization* and *Markov Random Fields* (MRFs) §3.6. While most of this material is covered in courses and textbooks on image processing, the use of optimization techniques is more typically associated with computer vision (although MRFs are now being widely used in image processing as well). The section on MRFs is also the first introduction to the use of Bayesian inference techniques, which are covered at a more abstract level in Appendix B. Chapter 3 also presents applications such as seamless image blending and image restoration.

Chapter 4 covers feature detection and matching. A lot of current 3D reconstruction and recognition techniques are built on extracting and matching *feature points* §4.1, so this is a fundamental technique required by many subsequent chapters (§5, §6, §8 and §14). I also cover edge and straight line detection §4.2–4.3, contour detection and tracking §4.4 and region segmentation §4.5. All of these techniques are essential building blocks that are widely used in a variety of applications, including performance-driven animation and interactive image editing.

Chapter 5 covers geometric alignment and camera calibration. I introduce the basic techniques of feature-based alignment in §5.1, and show how this problem can be solved using either linear or non-linear least squares, depending on the motion involved. I also introduce additional concept such as uncertainty weighting and robust regression, which are essential to making real-world systems work. Feature-based alignment is then used as a building block for 3D pose estimation (*extrinsic calibration*) §5.2 and camera (*intrinsic*) calibration §5.3. Chapter 5 also describes applications of these techniques to photo alignment for flip-book animations, 3D pose estimation from a hand-held camera, and single-view reconstruction of building models.

Chapter 6 covers the topic of *structure from motion*, which involves the simultaneous recovery of 3D camera motion and 3D scene structure from a collection of tracked 2D features. This chapter begins with the easier problem of 3D point *triangulation* §6.1, which is the 3D reconstruction of points from matched features when the camera positions are known. It then describes two-frame structure from motion §6.2, for which algebraic techniques exist, as well as robust sampling techniques such as RANSAC that can discount erroneous feature matches. The second half of Chapter 6 describes techniques for multi-frame structure from motion, including factorization §6.3, bundle adjustment §6.4, and constrained motion and structure models §6.5. It also presents applications in view morphing, sparse 3D model construction, and match move.

In Chapter 7, I go back to a topic that directly deals with image intensities (as opposed to feature tracks), namely dense intensity-based motion estimation (*optical flow*). I start with the

simplest possible motion models, namely translational motion §7.1, and cover topics such as hierarchical (coarse-to-fine) motion estimation, Fourier-based techniques, and iterative refinement. I then present parametric motion models, which can be used to compensate for camera rotation and zooming, as well as affine or planar perspective motion §7.2. This is then generalized to spline-based motion models §7.3 and then finally to general per-pixel optical flow §7.4, including layered and learned motion models §7.5–7.6. Applications of these techniques include automated morphing, frame interpolation (slow motion), and motion-based user interfaces.

Chapter 8 is devoted to *image stitching*, i.e., the construction of large panoramas and composites. While stitching is just one example of *computation photography* (§9), there is enough depth here to warrant its own chapter. I start by discussing various possible motion models §8.1, including planar motion and pure camera rotation. I then discuss global alignment §8.2, which is a special (simplified) case of general bundle adjustment, and then present *panorama recognition*, i.e., techniques for automatically discovering which images actually form overlapping panoramas. Finally, I cover the topics of *image compositing* and *blending* §8.3, which involve both selecting which pixels get used from which images, and blending them together so as to disguise exposure differences.

Image stitching is a wonderful application that ties together most of the material covered in earlier parts of this book. It also makes a good mid-term course project that can build on previously developed techniques such as image warping and feature detection and matching. Chapter 8 also presents more specialized variants of stitching such as whiteboard and document scanning, video summarization, *panography*, full 360° spherical panoramas, and interactive photomontage for blending repeated action shots together.

Chapter 9 presents additional examples of *computational photography*, which is the process of creating novel images from one or more input photographs, often based on the careful modeling and calibration of the image formation process §9.1. Computational photography techniques include merging multiple exposures to create *high dynamic range* images §9.2, increasing image resolution through blur removal and *super-resolution* §9.3, and image editing and compositing operations §9.4. I also cover the topics of texture analysis, synthesis and *inpainting* (hole filling) §9.5, as well as non-photorealistic rendering §9.6.

In chapter 10, I turn to the issue of stereo correspondence, which can be thought of as a special case of motion estimation where the camera positions are already known §10.1. This additional knowledge enables stereo algorithms to search over a much smaller space of correspondences, and in many cases to produce dense depth estimates that can be converted into visible surface models §10.3. I also cover multi-view stereo algorithms that build a true 3D surface representation instead of just a single depth map §10.6. Applications of stereo matching include head and gaze tracking, as well as depth-based background replacement (*Z-keying*).

Chapter 11 covers additional 3D shape and appearance modeling techniques. These include

classic *shape-from-X* techniques such as shape from shading, shape from texture, and shape from focus §11.1, as well as shape from smooth occluding contours §11.2 and silhouettes §11.3. An alternative to all of these *passive* computer vision techniques is to use *active rangefinding* §11.4, i.e., to project patterned light onto the scenes and to recover the 3D geometry through triangulation. Processing all of these 3D representations often involves interpolating and/or simplifying the geometry §11.5, or using alternative representations such as surface point sets §11.6.

The collection of techniques for going from one or more images to partial or full 3D models is often called *image-based modeling* or *3D photography*. The next part of Chapter 11 examines three more specialized application areas (architecture, faces, and human bodies), which can use *model-based reconstruction* to fit parameterized models to the sensed data §11.7. The final part of Chapter 11 examines the topic of *appearance modeling* §11.8, i.e., techniques for estimating the texture maps, albedos, or even sometimes complete *bi-directional reflectance distribution functions* (BRDFs) that describe the appearance of 3D surfaces.

In Chapter 12, I discuss the large number of image-based rendering techniques that have been developed in the last decade, including simpler techniques such as view interpolation §12.1, layered depth images §12.2, and sprites and layers §12.2.2, as well as the more general framework of lightfields and Lumigraphs §12.3 and higher-order fields such as environment mattes §12.4. Applications of these techniques include navigating 3D collections of photographs using *Photo Tourism* and viewing 3D models as *object movies*.

Chapter 13 discusses video-based rendering, which is the temporal extension of image-based rendering. The topics I cover include video enhancement and visual effects §13.1, video-based facial animation §13.3, periodic video turned into *video textures* §13.4, and 3D video constructed from multiple video streams §13.5. Applications of these techniques include video denoising, morphing, and walkthroughs/tours based on 360° video.

Chapter 14 describes different approaches to recognition. It begins with techniques for detecting and recognizing faces §14.1–§14.2, then looks at techniques for finding and recognizing particular objects (*instance recognition*) §14.3. Next, I cover the most difficult variant of recognition, namely the recognition of broad *categories*, such as cars, motorcycles, horses, and other animals §14.4 and the role that scene context plays in recognition §14.5.

The book also contains three appendices with more detailed mathematical topics and additional material. Appendix A covers linear algebra and numerical techniques, including matrix algebra, least-squares, and iterative techniques. Appendix B covers Bayesian estimation theory, including maximum likelihood, robust statistics, Markov Random Fields, and uncertainty modeling. Appendix C describes the supplementary material available to complement this book, including images and data sets, pointers to software, course slides, and an on-line bibliography.

Week	Material	Project
(1.)	§2 Image formation	P1
2.	§3 Image processing	
3.	§4 Feature detection and matching	
4.	§5 Projective geometry and alignment	
5.	§8 Image stitching and blending	
6.	§7 Optical flow and tracking	
7.	§6 Structure from motion	
8.	§14 Recognition	
(9.)	§9 More computational photography	
10.	§10 Stereo matching	
(11.)	§11 Multi-view stereo and 3D modeling	PP
12.	§12 Image-based rendering	
13.	Final project presentations	

Table 1.1: *Sample syllabi for 10-week and 13-week courses. The weeks in parenthesis are not used in the shorter 10-week version. P1 and P2 are two early-term mini-projects, PP is when the (student selected) final project proposals are due, and FP is the final project presentations.*

Sample syllabus

Teaching all of the material covered in this book in a single quarter or semester course is a Herculean task, and likely one not worth attempting. It is better to simply pick and choose topics related to the lecturer’s preferred emphasis, as well as tailored to the set of mini-projects envisioned for the students.

Steve Seitz and I have successfully used a 10-week syllabus similar to the one shown in Table 1.1 (omitting the parenthesized weeks) as both an undergraduate and a graduate-level course in computer vision. The undergraduate course⁹ tends to go lighter on the mathematics and takes more time reviewing basics, while the graduate level course¹⁰ dives more deeply into techniques and assumes the students already have a decent grounding in either vision or related mathematical techniques. (See also the *Introduction to Computer Vision* course at Stanford¹¹, which uses a similar curriculum.) Related courses have also been taught on the topics of 3D Photography¹² and Computational Photography¹³.

⁹ <http://www.cs.washington.edu/education/courses/455/>

¹⁰ <http://www.cs.washington.edu/education/courses/576/>

¹¹ <http://cs223b.stanford.edu/>

¹² <http://www.cs.washington.edu/education/courses/558/06sp/>

¹³ <http://graphics.cs.cmu.edu/courses/15-463/>

When Steve and I teach the course, we prefer to give the students several small programming projects early in the course rather than focusing on written homework and/or quizzes. With a suitable choice of topics, it is possible for these projects to build on each other. For example, introducing feature matching early on can be used in a second assignment to do image alignment and stitching. Alternatively, direct (optical flow) techniques can be used to do the alignment, and more focus can be put on either graph cut seam selection or multi-resolution blending techniques.

We also ask the students to propose a final project (with a set of suggested topics for those who need ideas) by the middle of the course, and reserve the last week of the class for student presentations. With any luck, some of these final projects can actually turn into conference submissions!

No matter how you decide to structure the course, or how you choose to use this book, I encourage you to try at least a few small programming tasks to get a good feel for how vision techniques work, and when they do not. Better yet, pick topics that are fun and can be used on your own photographs, and try to push your creative boundaries to come up with surprising results.

A note on notation

For better or worse, the notation found in computer vision and multi-view geometry textbooks tends to vary all over the map (Faugeras 1993, Hartley and Zisserman 2004, Girod *et al.* 2000, Faugeras and Luong 2001, Forsyth and Ponce 2003). In this book, I use the convention I first learned in my high school physics class (and later multi-variate calculus and computer graphics courses), which is that vectors v are lower case bold, matrices M are upper case bold, and scalars (T, s) are mixed case italic. Unless otherwise noted, vectors operate as column vector, i.e., they post-multiply matrices, Mv . Some commonly used matrices are R for rotations, K for calibration matrices, and I for the identity. The List of Symbols page right after the Preface contains a list of the most commonly used symbols in this book.

[Note: Could also move this whole paragraph onto that page.]

[Note: Can also talk about the following:

For succinctness, vectors are sometime written as comma-separated parenthesized lists $x = (x, y)$ instead of bracketed column vectors $x = [x \ y]^T$.

For homogeneous coordinates (§2.1), use $\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{x}$ in \mathcal{P}^2 .

Cross product operator in matrix form is denoted by $[]_\times$.]

Reference (background) material

[Note: Skip this section? I review suggested references in each individual chapter...]

This book attempts to be self-contained, so that students can implement the basic assignments and algorithms described here without the need for outside references. However, it does pre-

suppose a general familiarity with basic concepts in linear algebra and numerical techniques, which are reviewed in Appendix A, and image processing, which is reviewed in Chapter 3.

Students who want to delve in more deeply into these topics can look in (Golub and Van Loan 1996) for matrix algebra and (Strang 1988) for linear algebra. In image processing, there are a number of popular references, including (Crane 1997, Gomes and Velho 1997, Jähne 1997, Pratt 2001, Gonzales and Woods 2002, Russ 2007). For computer graphics, popular texts include (Foley *et al.* 1995, Watt 1995), with (Glassner 1995) providing a more in-depth look at image formation and rendering. For statistics and machine learning, Chris Bishop's (2006) book is a wonderful and comprehensive introduction with a wealth of exercises. Students may also want to look in other textbooks on computer vision for material that I do not cover here, as well as for additional project ideas (Ballard and Brown 1982, Faugeras 1993, Nalwa 1993, Trucco and Verri 1998, Forsyth and Ponce 2003).

There is, however, no substitute for reading the latest research literature, both for the latest ideas and techniques, and for the most up-to-date references to related literature.¹⁴ In this book, I have attempted to cite the most recent work in each field so that students can read these directly and use them as inspiration for their own work. Browsing the last few years' conference proceedings of the major graphics and vision conferences such as SIGGRAPH, CVPR, ECCV, and ICCV will provide a wealth of new ideas. The tutorials offered at these conferences, for which slides and/or excellent notes are often available, are also an invaluable resource.

¹⁴ For a comprehensive bibliography and taxonomy of computer vision research, Keith Price's Annotated Computer Vision Bibliography <http://iris.usc.edu/Vision-Notes/bibliography/contents.html> is an invaluable resource.

Chapter 2

Image formation

2.1	Geometric image formation	32
2.1.1	Geometric primitives	32
2.1.2	2D transformations	35
2.1.3	3D transformations	39
2.1.4	3D to 2D projections	46
2.1.5	Lens distortions	58
2.2	Photometric image formation	60
2.2.1	Lighting	60
2.2.2	Reflectance and shading	61
2.2.3	Optics	68
2.3	The digital camera	73
2.3.1	Sampling and aliasing	77
2.3.2	Color	80
2.3.3	Compression	91
2.4	Exercises	92

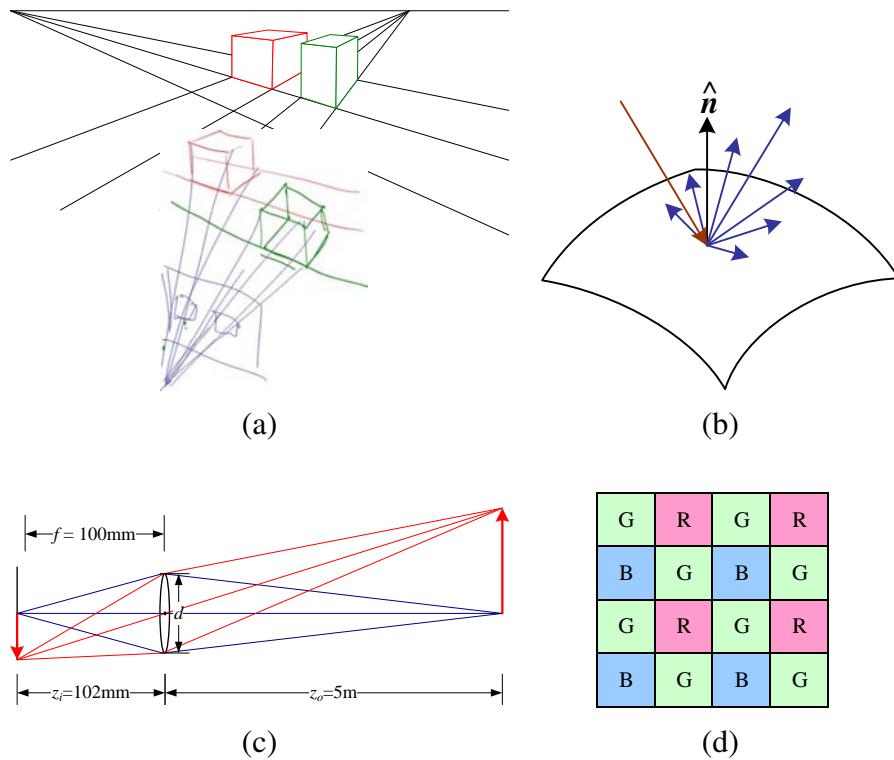


Figure 2.1: *A few components of the image formation process: (a) perspective projection; (b) light scattering when hitting a surface; (c) lens optics; (d) Bayer color filter array.*

[Note: Steve Seitz has suggested that this material be moved later in the book. In particular, the geometric image formation is not really needed until the Calibration chapter §5, and could form its own chapter call Projective Geometry. (But, line equations are useful in the feature detection section §4.3.) Photometric image formation is needed in the computational photography Chapter §9 and also in the Shape-from-X §11.1 and albedo recovery §11.8 sections. Notions of sampling and aliasing are needed when discussing interpolation and decimation filters in §3.4.1.

My preference is to keep all this material up front, and to tell the reader to skip ahead if they already know about graphics and/or are not that interested in how images are exactly formed.]

Before we can intelligently analyze and manipulate images, we need to establish a vocabulary for describing the geometry of a scene. We also need to understand the image formation process that produced a particular image given a set of lighting conditions, scene geometry, surface properties, and camera optics. In this chapter, I present a simplified model of such an image formation process. Section §2.1 introduces the basic geometric primitives used throughout the book (points, lines, and planes) and the *geometric* transformations that project these 3D quantities into 2D image features (Figure 2.1a). Section §2.2 describes how lighting, surface properties (Figure 2.1b), and camera *optics* (Figure 2.1c) interact in order to produce the color values that fall onto the image sensor. Section §2.3 describes how continuous color images are turned into discrete digital *samples* inside the image sensor (Figure 2.1d) and how to avoid (or at least characterize) sampling deficiencies such as aliasing.

The material covered in this chapter is but a brief summary of a very rich and deep set of topics, traditionally covered in a number of separate fields. A more thorough introduction to the geometry of points, lines, planes, and projections can be found in textbooks on multi-view geometry (Hartley and Zisserman 2004, Faugeras and Luong 2001) and computer graphics (Foley *et al.* 1995). The image formation (synthesis) process is traditionally taught as part of a computer graphics curriculum (Foley *et al.* 1995, Glassner 1995, Watt 1995, Shirley 2005), but it is also studied in physics-based computer vision (Wolff *et al.* 1992a). The behavior of camera lens systems is studied in optics (Möller 1988, Hecht 2001, Ray 2002). Two good books on color theory are (Wyszecki and Stiles 1982, Healey and Shafer 1992). Finally, topics relating to sampling and aliasing are covered in textbooks on signal and image processing (Crane 1997, Jähne 1997, Oppenheim and Schafer 1996, Oppenheim *et al.* 1999, Pratt 2001).

A note to students: If you have already studied computer graphics, you may want to skim the material in the section on geometric image formation §2.1, although the sections on projective depth and object-centered projection near the end of §2.1.4 may be new to you. Similarly, physics students (as well as computer graphics students) will mostly be familiar with the section on photometric image formation §2.2. Finally, students with a good background in image processing will already be familiar with sampling issues §2.3 as well as some of the material in the next chapter on image processing.

2.1 Geometric image formation

In this section, I introduce the basic 2D and 3D primitives used in this textbook, namely points, lines, and planes. I also describe how 3D features are projected into 2D features. More detailed descriptions of these topics (along with a gentler and more intuitive introduction) can be found in textbooks on multiple-view geometry (Hartley and Zisserman 2004, Faugeras and Luong 2001).

2.1.1 Geometric primitives

Geometric primitives form the basic building blocks used to describe three-dimensional shape. In this section, I introduce points, lines, and planes. Later sections of the book cover curves (§4.4 and §11.2), surfaces (§11.5), and volumes (§11.3).

2D Points. 2D Points (pixel coordinates in an image) can be denoted using a pair of values, $\mathbf{x} = (x, y) \in \mathcal{R}^2$, or alternatively,

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}. \quad (2.1)$$

(As stated in the introduction, we use the (x_1, x_2, \dots) notation to denote column vectors.)

2D points can also be represented using *homogeneous coordinates*, $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$, where vectors that differ only by scale are considered to be equivalent. $\mathcal{P}^2 = \mathcal{R}^3 - (0, 0, 0)$ is called the 2D *projective space*.

A homogeneous vector $\tilde{\mathbf{x}}$ can be converted back into an *inhomogeneous* vector \mathbf{x} by dividing through by the last element \tilde{w} , i.e.,

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{\mathbf{x}}, \quad (2.2)$$

where $\bar{\mathbf{x}} = (x, y, 1)$ is the *augmented vector*. Homogeneous points whose last element is $\tilde{w} = 0$ are called *ideal points* or *points at infinity* and do not have an equivalent inhomogeneous representation.

2D Lines. 2D lines can also be represented using homogeneous coordinates $\tilde{\mathbf{l}} = (a, b, c)$. The corresponding *line equation* is

$$\bar{\mathbf{x}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0. \quad (2.3)$$

We can normalize the line equation vector so that $\mathbf{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\mathbf{n}}, d)$ with $\|\hat{\mathbf{n}}\| = 1$. In this case, $\hat{\mathbf{n}}$ is the *normal vector* perpendicular to the line, and d is its distance to the origin (Figure 2.2). (The one exception to this normalization is the *line at infinity* $\tilde{\mathbf{l}} = (0, 0, 1)$, which includes all (ideal) points at infinity.)

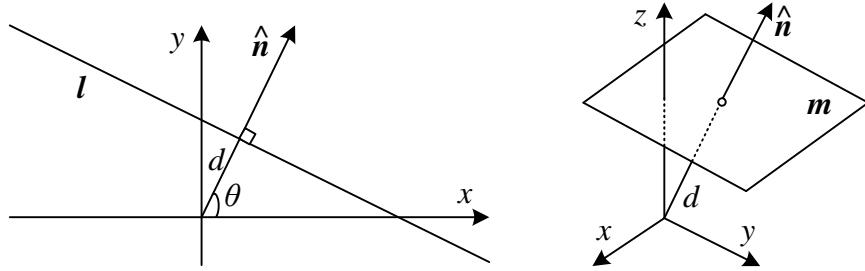


Figure 2.2: 2D line equation and 3D plane equation, expressed in terms of the normal $\hat{\mathbf{n}}$ and distance to the origin d .

We can also express $\hat{\mathbf{n}}$ as a function of rotation angle θ , $\hat{\mathbf{n}} = (\hat{n}_x, \hat{n}_y) = (\cos \theta, \sin \theta)$ (Figure 2.2). This representation is commonly used in the *Hough transform* line finding algorithm, which is discussed in §4.3.2. The combination (θ, d) is also known as *polar coordinates*.

When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2, \quad (2.4)$$

where \times is the cross-product operator. Similarly, the line joining two points can be written as

$$\tilde{\mathbf{l}} = \tilde{\mathbf{x}}_1 \times \tilde{\mathbf{x}}_2. \quad (2.5)$$

When trying to fit an intersection point to multiple lines, or conversely a line to multiple points, least square techniques (§5.1.1 and Appendix A.2) can be used, as discussed in Exercise 2.1.

2D Conics. There are also other algebraic curves that can be expressed with simple polynomial homogeneous equations. For example, the *conic sections* (so called because they arise as the intersection of a plane and a 3D cone) can be written using a *quadratic* equation

$$\tilde{\mathbf{x}}^T \mathbf{Q} \tilde{\mathbf{x}} = 0. \quad (2.6)$$

These play useful roles in the study of multi-view geometry and camera calibration (Hartley and Zisserman 2004, Faugeras and Luong 2001) but are not used extensively in this book.

3D Points. Point coordinates in three dimensions can be written using inhomogeneous coordinates $\mathbf{x} = (x, y, z) \in \mathcal{R}^3$ or homogeneous coordinates $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$. As before, it is sometimes useful to denote a 3D point using the augmented vector $\bar{\mathbf{x}} = (x, y, z, 1)$ with $\tilde{\mathbf{x}} = \tilde{w}\bar{\mathbf{x}}$.

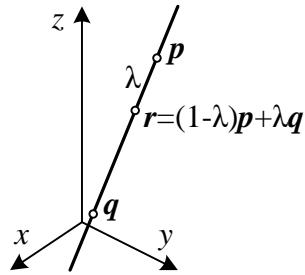


Figure 2.3: 3D line equation $r = (1 - \lambda)p + \lambda q$.

3D Planes. 3D planes can also be represented as homogeneous coordinates $\tilde{\mathbf{m}} = (a, b, c, d)$ with a corresponding plane equation

$$\bar{x} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0. \quad (2.7)$$

We can also normalize the plane equation as $\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\mathbf{n}}, d)$ with $\|\hat{\mathbf{n}}\| = 1$. In this case $\hat{\mathbf{n}}$ is the *normal vector* perpendicular to the plane, and d is its distance to the origin (Figure 2.2). As with the case of 2D lines, the *plane at infinity* $\tilde{\mathbf{m}} = (0, 0, 0, 1)$, which contains all the points at infinity, cannot be normalized (i.e., it does not have a unique normal, nor does it have a finite distance).

We can express $\hat{\mathbf{n}}$ as a function of two angles (θ, ϕ) ,

$$\hat{\mathbf{n}} = (\cos \theta \cos \phi, \sin \theta \cos \phi, \sin \phi), \quad (2.8)$$

i.e., using *spherical coordinates*, but these are less commonly used than polar coordinates since they do not uniformly sample the space of possible normal vectors.

3D Lines. Lines in 3D are less elegant than either lines in 2D or planes in 3D. One possible representation is to use two points on the line, (p, q) . Any other point on the line can be expressed as a linear combination of these two points

$$\mathbf{r} = (1 - \lambda)p + \lambda q, \quad (2.9)$$

as shown in Figure 2.3. If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining p and q .

If we use homogeneous coordinates, we can write the line as

$$\tilde{\mathbf{r}} = \mu \tilde{\mathbf{p}} + \lambda \tilde{\mathbf{q}}. \quad (2.10)$$

A special case of this is when the second point is at infinity, i.e., $\tilde{\mathbf{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\mathbf{d}}, 0)$. Here, we see that $\hat{\mathbf{d}}$ is the *direction* of the line. We can then re-write the inhomogeneous 3D line equation as

$$\mathbf{r} = \mathbf{p} + \lambda \hat{\mathbf{d}}. \quad (2.11)$$

A disadvantage of the endpoint representation for 3D lines is that it has too many degrees of freedom, i.e., 6 (3 for each endpoint), instead of the 4 degrees that a 3D line truly has. However, if we fix the two points on line to lie in specific planes, we obtain a 4 d.o.f. representation. For example, if we are representing nearly vertical lines, then $z = 0$ and $z = 1$ form two suitable planes, i.e., the (x, y) coordinates in both planes provide the 4 coordinates describing the line. This kind of two-plane parameterization is used in the *Lightfield* and *Lumigraph* image-based rendering systems described in §12 to represent the collection of rays seen by a camera as it moves in front of an object. The two endpoint representation is also useful for representing line segments, even when their exact endpoints cannot be seen (only guessed at).

If we wish to represent all possible lines without bias towards any particular orientation, we can use *Plücker coordinates* (Hartley and Zisserman 2004, Chapter 2)(Faugeras and Luong 2001, Chapter 3). These coordinates are the six independent non-zero entries in the 4×4 skew symmetric matrix

$$\mathbf{L} = \tilde{\mathbf{p}}\tilde{\mathbf{q}}^T - \tilde{\mathbf{q}}\tilde{\mathbf{p}}^T, \quad (2.12)$$

where $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ are *any* two (non-identical) points on the line. This representation has only 4 degrees of freedom, since \mathbf{L} is homogeneous and also satisfies $\det(\mathbf{L}) = 0$, which results in a quadratic constraint on the Plücker coordinates.

In practice, I find that in most applications, the minimal representation is not essential. An adequate model of 3D lines can be obtained by estimating their direction (which may be known ahead of time, e.g., for architecture) and some point within the visible portion of the line (e.g., see §6.5.1), or by using the two endpoints, since lines are most often visible as finite line segments. However, if you are interested in more details about the topic of minimal line parameterizations, Förstner (2005) discusses various ways to infer and model 3D lines in projective geometry, as well as how to estimate the uncertainty in such fitted models.

3D Quadrics. The 3D analogue to conic sections are quadric surfaces

$$\bar{\mathbf{x}}^T \mathbf{Q} \bar{\mathbf{x}} = 0 \quad (2.13)$$

(Hartley and Zisserman 2004, Chapter 2). Again, while these are useful in the study of multi-view geometry and can also serve as useful modeling primitives (spheres, ellipsoids, cylinders), we do not study them in great detail in this book.

2.1.2 2D transformations

Having defined our basic primitives, we can now turn our attention to how they can be transformed. The simplest transformations occur in the 2D plane and are illustrated in Figure 2.4.

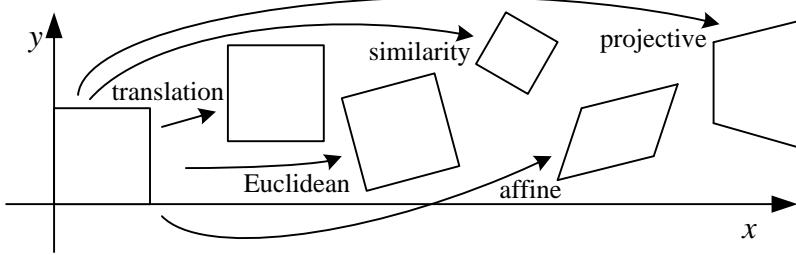


Figure 2.4: Basic set of 2D planar transformations

Translation. 2D translations can be written as $\mathbf{x}' = \mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = [\mathbf{I} \quad \mathbf{t}] \bar{\mathbf{x}} \quad (2.14)$$

where \mathbf{I} is the (2×2) identity matrix or

$$\bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \bar{\mathbf{x}} \quad (2.15)$$

where $\mathbf{0}$ is the zero vector. Using a 2×3 matrix results in a more compact notation, whereas using a full-rank 3×3 matrix (which can be obtained from the 2×3 matrix by appending a $[0^T \ 1]$ row) makes it possible to chain transformations together using matrix multiplication. Note that in any equation where an augmented vector such as $\bar{\mathbf{x}}$ appears on both sides, it can always be replaced with a full homogeneous vector $\tilde{\mathbf{x}}$.

geometric transformations!rigid body

Rotation + translation. This transformation is also known as *2D rigid body motion* or the *2D Euclidean transformation* (since Euclidean distances are preserved). It can be written as $\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = [\mathbf{R} \quad \mathbf{t}] \bar{\mathbf{x}} \quad (2.16)$$

where

$$\mathbf{R} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (2.17)$$

is an orthonormal rotation matrix with $\mathbf{R}\mathbf{R}^T = \mathbf{I}$ and $|\mathbf{R}| = 1$.

Scaled rotation. Also known as the *similarity transform*, this transform can be expressed as $\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t}$ where s is an arbitrary scale factor. It can also be written as

$$\mathbf{x}' = [s\mathbf{R} \quad \mathbf{t}] \bar{\mathbf{x}} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{\mathbf{x}}, \quad (2.18)$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

Affine. The affine transform is written as $\mathbf{x}' = \mathbf{A}\bar{\mathbf{x}}$, where \mathbf{A} is an arbitrary 2×3 matrix, i.e.,

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{\mathbf{x}}. \quad (2.19)$$

Parallel lines remain parallel under affine transformations.

Projective. This transform, also known as a *perspective transform* or *homography*, operates on homogeneous coordinates,

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}, \quad (2.20)$$

where $\tilde{\mathbf{H}}$ is an arbitrary 3×3 matrix. Note that $\tilde{\mathbf{H}}$ is itself homogeneous, i.e., it is only defined up to a scale, and that two $\tilde{\mathbf{H}}$ matrices that differ only by scale are equivalent. The resulting homogeneous coordinate $\tilde{\mathbf{x}}'$ must be normalized in order to obtain an inhomogeneous result \mathbf{x} , i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \quad (2.21)$$

Perspective transformations preserve straight lines (i.e., they remain straight after the transformation).

Hierarchy of 2D transformations. The preceding set of transformations are illustrated in Figure 2.4 and summarized in Table 2.1. The easiest way to think of these is as a set of (potentially restricted) 3×3 matrices operating on 2D homogeneous coordinate vectors. Hartley and Zisserman (2004) contains a more detailed description of the hierarchy of 2D planar transformations.

The above transformations form a nested set of *groups*, i.e., they are closed under composition and have an inverse that is a member of the same group. (This will be important later when applying these transformations to images in §3.5.) Each (simpler) group is a subset of the more complex group below it.

Co-vectors. While the above transformations can be used to transform points in a 2D plane, can they also be used directly to transform a line equation? Consider the homogeneous equation $\tilde{\mathbf{l}} \cdot \tilde{\mathbf{x}} = 0$. If we transform $\mathbf{x}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$, we obtain

$$\tilde{\mathbf{l}} \cdot \tilde{\mathbf{x}}' = \tilde{\mathbf{l}}^T \tilde{\mathbf{H}} \tilde{\mathbf{x}} = (\tilde{\mathbf{H}}^T \tilde{\mathbf{l}})^T \tilde{\mathbf{x}} = \tilde{\mathbf{l}} \cdot \tilde{\mathbf{x}} = 0, \quad (2.22)$$

i.e., $\tilde{\mathbf{l}} = \tilde{\mathbf{H}}^{-T} \tilde{\mathbf{l}}$. Thus, the action of a projective transformation on a *co-vector* such as a 2D line or 3D normal can be represented by the transposed inverse of the matrix, which is equivalent to

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation + ⋯	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths + ⋯	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles + ⋯	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism + ⋯	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

Table 2.1: *Hierarchy of 2D coordinate transformations. The 2×3 matrices are extended with a third $[0^T \ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.*

the *adjoint* of $\tilde{\mathbf{H}}$, since projective transformation matrices are homogeneous. Jim Blinn's (1998) book *Dirty Pixels* contains two chapters (9 and 10) describing the ins and outs of notating and manipulating co-vectors.

Additional transformations.

While the above transformations are the ones we use most extensively, a number of additional transformations are sometimes used.

Stretch/squash. This transformation changes the aspect ratio of an image,

$$\begin{aligned} x' &= s_x x + t_x \\ y' &= s_y y + t_y, \end{aligned}$$

and is a restricted form of an affine transformation. Unfortunately, it does not nest cleanly with the groups listed in Table 2.1.

Planar surface flow. This 8-parameter transformation (Horn 1986, Bergen *et al.* 1992, Girod *et al.* 2000),

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_6 x^2 + a_7 x y \\ y' &= a_3 + a_4 x + a_5 y + a_7 x^2 + a_6 x y, \end{aligned}$$

arises when a planar surface undergoes a small 3D motion. It can thus be thought of as a small motion approximation to a full homography. Its main attraction is that it is *linear* in the motion parameters a_k , which are often the quantities being estimated.

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{3 \times 4}$	3	orientation + ⋯	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{3 \times 4}$	6	lengths + ⋯	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{3 \times 4}$	7	angles + ⋯	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{3 \times 4}$	12	parallelism + ⋯	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{4 \times 4}$	15	straight lines	

Table 2.2: *Hierarchy of 3D coordinate transformations. The 3×4 matrices are extended with a third $[0^T \ 1]$ row to form a full 4×4 matrix for homogeneous coordinate transformations. The mnemonic icons are only drawn in 2D, but are meant to suggest transformations occurring in a full 3D cube.*

Bilinear interpolant. This 8-parameter transform (Wolberg 1990),

$$\begin{aligned} x' &= a_0 + a_1x + a_2y + a_6xy \\ y' &= a_3 + a_4x + a_5y + a_7xy, \end{aligned}$$

can be used to interpolate the deformation due to the motion of the four corner points of a square. (In fact, it can interpolate the motion of any four non-collinear points.) While the deformation is linear in the motion parameters, it does not generally preserve straight lines (only lines parallel to the square axes). However, it is often quite useful, e.g., in the interpolation of sparse grids using splines §7.3.

2.1.3 3D transformations

The set of three-dimensional coordinate transformations is very similar to that available for 2D transformations, and is summarized in Table 2.2. As in 2D, these transformations for a nested set of groups. (See (Hartley and Zisserman 2004, §2.4) for a more detailed description of this hierarchy.)

Translation. 3D translations can be written as $\mathbf{x}' = \mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}} \quad (2.23)$$

where \mathbf{I} is the (3×3) identity matrix and $\mathbf{0}$ is the zero vector.

Rotation + translation. Also known as 3D *rigid body motion* or the 3D *Euclidean transformation*, it can be written as $\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}$ or

$$\mathbf{x}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}} \quad (2.24)$$

where \mathbf{R} is a 3×3 orthonormal rotation matrix with $\mathbf{R}\mathbf{R}^T = \mathbf{I}$ and $|\mathbf{R}| = 1$. Note that sometimes it is more convenient to describe a rigid motion using

$$\mathbf{x}' = \mathbf{R}(\mathbf{x} - \mathbf{c}) = \mathbf{R}\mathbf{x} - \mathbf{R}\mathbf{c}, \quad (2.25)$$

where \mathbf{c} is the center of rotation (often the camera center).

Compactly parameterizing a 3D rotation is a non-trivial task, which we describe in more detail below.

Scaled rotation. The 3D *similarity transform* can be expressed as $\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t}$ where s is an arbitrary scale factor. It can also be written as

$$\mathbf{x}' = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}}. \quad (2.26)$$

This transform preserves angles between lines and planes.

Affine. The affine transform is written as $\mathbf{x}' = \mathbf{A}\bar{\mathbf{x}}$, where \mathbf{A} is an arbitrary 3×4 matrix, i.e.,

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix} \bar{\mathbf{x}}. \quad (2.27)$$

Parallel lines and planes remain parallel under affine transformations.

Projective. This transform, variously known as a 3D *perspective transform*, *homography*, or *collineation*, operates on homogeneous coordinates,

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}, \quad (2.28)$$

where $\tilde{\mathbf{H}}$ is an arbitrary 4×4 homogeneous matrix. As in 2D, the resulting homogeneous coordinate $\tilde{\mathbf{x}}'$ must be normalized in order to obtain an inhomogeneous result \mathbf{x} . Perspective transformations preserve straight lines (i.e., they remain straight after the transformation).

3D rotations

The biggest difference between 2D and 3D coordinate transformations is that the parameterization of the 3D rotation matrix \mathbf{R} is not as straightforward. Several possibilities exist:

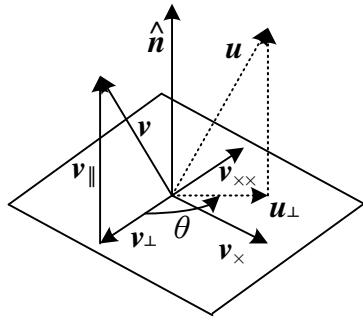


Figure 2.5: *Rotation around an axis \hat{n} by an angle θ . See the text for a description of the symbols.*

Euler angles

A rotation matrix can be formed as the product of three rotations around the cardinal x , y , and z axes. This is generally a bad idea, at the result depends on the order in which the transforms are applied. What's worse, it is not always possible to move smoothly in the parameter space, i.e., sometimes one or more of the Euler angles change dramatically in response to a small change in rotation. (In robotics, this is sometimes referred to as *gimbal lock*.) For these reasons, I do not even give the formula for Euler angles in this book. (Interested readers can look in other textbooks or technical reports, e.g., (Faugeras 1993, Diebel 2006).) Note that if in some applications, the rotations are known to be a set of uni-axial transforms, these can always be represented using an explicit set of rigid transformations.

Axis/angle (exponential twist)

[Note: Is the following too detailed? Can I leave it as an exercise? However, v_{\parallel} and v_{\perp} are also used in §2.2.2 Specular Reflection, so may need to leave this in.]

A rotation can be represented by a rotation axis \hat{n} and an angle θ , or equivalently by a 3D vector $\omega = \theta\hat{n}$. [Note: see if there's some way to fix up the boldface omega.] Figure 2.5 shows how we can compute the equivalent rotation. First, we project the vector v onto the axis \hat{n} to obtain

$$v_{\parallel} = \hat{n}(\hat{n} \cdot v) = (\hat{n}\hat{n}^T)v, \quad (2.29)$$

which is the component of v that is not affected by the rotation. Next, we compute the perpendicular residual of v from \hat{n} ,

$$v_{\perp} = v - v_{\parallel} = (I - \hat{n}\hat{n}^T)v. \quad (2.30)$$

We can rotate this vector by 90° using the cross product,

$$v_{\times} = \hat{n} \times v = [\hat{n}]_{\times}v, \quad (2.31)$$

where $[\hat{\mathbf{n}}]_{\times}$ is the matrix form of the cross-product operator with the vector $\hat{\mathbf{n}} = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$,

$$[\hat{\mathbf{n}}]_{\times} = \begin{bmatrix} 0 & -\hat{n}_z & \hat{n}_y \\ \hat{n}_z & 0 & -\hat{n}_x \\ -\hat{n}_y & \hat{n}_x & 0 \end{bmatrix}. \quad (2.32)$$

Note that rotating this vector by another 90° is equivalent to taking the cross product again,

$$\mathbf{v}_{\times\times} = \hat{\mathbf{n}} \times \mathbf{v}_{\times} = [\hat{\mathbf{n}}]_{\times}^2 \mathbf{v} = -\mathbf{v}_{\perp}$$

and hence

$$\mathbf{v}_{\parallel} = \mathbf{v} - \mathbf{v}_{\perp} = \mathbf{v} + \mathbf{v}_{\times\times} = (\mathbf{I} + [\hat{\mathbf{n}}]_{\times}^2) \mathbf{v}.$$

We can now compute the in-plane component of the rotated vector \mathbf{u} as

$$\mathbf{u}_{\perp} = \cos \theta \mathbf{v}_{\perp} + \sin \theta \mathbf{v}_{\times} = (\sin \theta [\hat{\mathbf{n}}]_{\times} - \cos \theta [\hat{\mathbf{n}}]_{\times}^2) \mathbf{v}.$$

Putting all these terms together, we obtain the final rotated vector as

$$\mathbf{u} = \mathbf{u}_{\perp} + \mathbf{v}_{\parallel} = (\mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_{\times} + (1 - \cos \theta) [\hat{\mathbf{n}}]_{\times}^2) \mathbf{v}. \quad (2.33)$$

We can therefore write the rotation matrix corresponding to a rotation by θ around an axis $\hat{\mathbf{n}}$ as

$$\mathbf{R}(\hat{\mathbf{n}}, \theta) = \mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_{\times} + (1 - \cos \theta) [\hat{\mathbf{n}}]_{\times}^2, \quad (2.34)$$

which is known as *Rodriguez's formula* (Ayache 1989).

The product of the axis $\hat{\mathbf{n}}$ and angle θ , $\omega = \theta \hat{\mathbf{n}} = (\omega_x, \omega_y, \omega_z)$, is a minimal representation for a 3D rotation. Rotations through common angles such as multiple of 90° can be represented exactly (and converted to exact matrices) if θ is stored in degrees. Unfortunately, this representation is not unique, since we can always add a multiple of 360° (2π radians) to θ and get the same rotation matrix. As well, $(\hat{\mathbf{n}}, \theta)$ and $(-\hat{\mathbf{n}}, -\theta)$ represent the same rotation.

However, for small rotations (e.g., corrections to rotations), this is an excellent choice. In particular, for small (infinitesimal or instantaneous) rotations and θ expressed in radians, Rodriguez's formula simplifies to

$$\mathbf{R}(\omega) \approx \mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_{\times} \approx \mathbf{I} + [\theta \hat{\mathbf{n}}]_{\times} = \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix}, \quad (2.35)$$

which gives a nice linearized relationships between the rotation parameters ω and \mathbf{R} . We can also write $\mathbf{R}(\omega)\mathbf{v} \approx \mathbf{v} + \omega \times \mathbf{v}$, which is handy when we want to compute the derivative of $\mathbf{R}\mathbf{v}$ with respect to ω ,

$$\frac{\partial \mathbf{R}\mathbf{v}}{\partial \omega^T} = -[\mathbf{v}]_{\times} = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}. \quad (2.36)$$

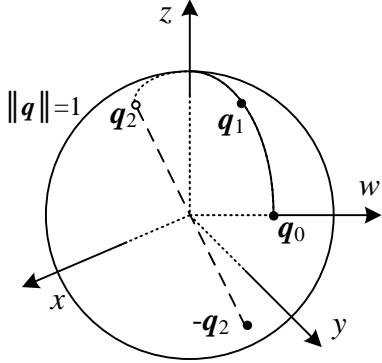


Figure 2.6: Unit quaternions live on the unit sphere $\|q\| = 1$. This figure shows a smooth trajectory through the three quaternions q_0 , q_1 , and q_2 . The antipodal point to q_2 , namely $-q_2$, represents the same rotation as q_2 .

Another way to derive a rotation through a finite angle is called the *exponential twist* (Murray *et al.* 1994). A rotation by an angle θ is equivalent to k rotations through θ/k . In the limit as $k \rightarrow \infty$, we obtain

$$\mathbf{R}(\hat{\mathbf{n}}, \theta) = \lim_{k \rightarrow \infty} (\mathbf{I} + \frac{1}{k} [\theta \hat{\mathbf{n}}]_{\times})^k = \exp [\omega]_{\times}. \quad (2.37)$$

If we expand the matrix exponential as a Taylor series (using the identity $[\hat{\mathbf{n}}]_{\times}^{k+2} = -[\hat{\mathbf{n}}]_{\times}^k$, $k > 0$, and again assuming θ is in radians),

$$\begin{aligned} \exp [\omega]_{\times} &= \mathbf{I} + \theta [\hat{\mathbf{n}}]_{\times} + \frac{\theta^2}{2} [\hat{\mathbf{n}}]_{\times}^2 + \frac{\theta^3}{3!} [\hat{\mathbf{n}}]_{\times}^3 + \dots \\ &= \mathbf{I} + (\theta - \frac{\theta^3}{3!} + \dots) [\hat{\mathbf{n}}]_{\times} + (\frac{\theta^2}{2} - \frac{\theta^3}{4!} + \dots) [\hat{\mathbf{n}}]_{\times}^2 \\ &= \mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_{\times} + (1 - \cos \theta) [\hat{\mathbf{n}}]_{\times}^2, \end{aligned} \quad (2.38)$$

which yields the familiar Rodriguez's formula.

Unit quaternions

The unit quaternion representation is closely related to the angle/axis representation. A unit quaternion is a unit length 4-vector whose components can be written as $\mathbf{q} = (q_x, q_y, q_z, q_w)$ or $\mathbf{q} = (x, y, z, w)$ for short. Unit quaternions live on the unit sphere $\|\mathbf{q}\| = 1$ and *antipodal* (opposite sign) quaternions, \mathbf{q} and $-\mathbf{q}$, represent the same rotation (Figure 2.6). Other than this ambiguity (dual covering), the unit quaternion representation of a rotation is unique. Furthermore, the representation is *continuous*, i.e., as rotation matrices vary continuously, one can find a continuous quaternion representation, although the path on the quaternion sphere may wrap all the way

around before returning to the “origin” $\mathbf{q}_o = (0, 0, 0, 1)$. For these and other reasons given below, quaternions are a very popular representation for pose and for pose interpolation in computer graphics ([Shoemake 1985](#)).

Quaternions can be derived from the axis/angle representation through the formula

$$\mathbf{q} = (\mathbf{v}, w) = \left(\sin \frac{\theta}{2} \hat{\mathbf{n}}, \cos \frac{\theta}{2} \right), \quad (2.39)$$

where $\hat{\mathbf{n}}$ and θ are the rotation axis and angle. Using the trigonometric identities $\sin \theta = 2 \sin \frac{\theta}{2} \cos \frac{\theta}{2}$ and $(1 - \cos \theta) = 2 \sin^2 \frac{\theta}{2}$, Rodriguez’s formula can be converted to

$$\begin{aligned} \mathbf{R}(\hat{\mathbf{n}}, \theta) &= \mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_{\times} + (1 - \cos \theta) [\hat{\mathbf{n}}]_{\times}^2 \\ &= \mathbf{I} + 2w[\mathbf{v}]_{\times} + 2[\mathbf{v}]_{\times}^2. \end{aligned} \quad (2.40)$$

This suggests a quick way to rotate a vector \mathbf{v} by a quaternion using a series of cross products, scalings, and additions. To obtain a formula for $\mathbf{R}(\mathbf{q})$ as a function of (x, y, z, w) , recall that

$$[\mathbf{v}]_{\times} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \text{ and } [\mathbf{v}]_{\times}^2 = \begin{bmatrix} -y^2 - z^2 & xy & xz \\ xy & -x^2 - z^2 & yz \\ xz & yz & -x^2 - y^2 \end{bmatrix}.$$

We thus obtain

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - zw) & 2(xz + yw) \\ 2(xy + zw) & 1 - 2(x^2 + z^2) & 2(yz - xw) \\ 2(xz - yw) & 2(yz + xw) & 1 - 2(x^2 + y^2) \end{bmatrix}. \quad (2.41)$$

The diagonal terms can be made more symmetrical by replacing $1 - 2(y^2 + z^2)$ with $(x^2 + w^2 - y^2 - z^2)$, etc.

The nicest aspect of unit quaternions is that there is a simple algebra for composing rotations expressed as unit quaternions. Given two quaternions $\mathbf{q}_0 = (\mathbf{v}_0, w_0)$ and $\mathbf{q}_1 = (\mathbf{v}_1, w_1)$, the *quaternion multiply* operator is defined as

$$\mathbf{q}_2 = \mathbf{q}_0 \mathbf{q}_1 = (\mathbf{v}_0 \times \mathbf{v}_1 + w_0 \mathbf{v}_1 + w_1 \mathbf{v}_0, w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1), \quad (2.42)$$

with the property that $\mathbf{R}(\mathbf{q}_2) = \mathbf{R}(\mathbf{q}_0) \mathbf{R}(\mathbf{q}_1)$. Note that quaternion multiplication is *not* commutative, just as 3D rotations and matrix multiplications are not.

Taking the inverse of a quaternion is easy: just flip the sign of \mathbf{v} or w (but not both!). (You can verify this has the desired effect of transposing the \mathbf{R} matrix in (2.41).) Thus, we can also define *quaternion division* as

$$\mathbf{q}_2 = \mathbf{q}_0 / \mathbf{q}_1 = \mathbf{q}_0 \mathbf{q}_1^{-1} = (\mathbf{v}_0 \times \mathbf{v}_1 + w_0 \mathbf{v}_1 - w_1 \mathbf{v}_0, -w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1). \quad (2.43)$$

This is useful when the *incremental rotation* between two rotations is desired.

In particular, if we want to determine a rotation that is partway between two given rotations, we can compute the incremental rotation, take a fraction of the angle, and compute the new rotation. This procedure is called *spherical linear interpolation* or *slerp* for short ([Shoemake 1985](#)), and can be written in more detail as:

procedure *slerp*($\mathbf{q}_0, \mathbf{q}_1, \alpha$):

1. $\mathbf{q}_r := \mathbf{q}_1 / \mathbf{q}_0 = (\mathbf{v}_r, w_r)$

2. if $w_r < 0$ then $\mathbf{q}_r := -\mathbf{q}_r$

3. $\theta_r := 2 \tan^{-1}(\|\mathbf{v}_r\|/w_r)$

4. $\hat{\mathbf{n}}_r := \mathcal{N}(\mathbf{v}_r) = \mathbf{v}_r / \|\mathbf{v}_r\|$

5. $\theta_\alpha := \alpha \theta_r$

6. $\mathbf{q}_\alpha := (\sin \frac{\theta_\alpha}{2} \hat{\mathbf{n}}_r, \cos \frac{\theta_\alpha}{2})$

7. $\mathbf{q}_2 := \mathbf{q}_\alpha \mathbf{q}_0$

endproc.

[Note: Do we want to keep this kind of pseudocode? Is it the only example in the book? Could make this an exercise as well.] Note that Shoemake's ([1985](#)) article presents two *different* formulas than the one given here. The first exponentiates \mathbf{q}_r by alpha before multiplying the original quaternion,

$$\mathbf{q}_2 = \mathbf{q}_r^\alpha \mathbf{q}_0, \quad (2.44)$$

while the second treat the quaternions as 4-vectors on a sphere and uses

$$\mathbf{q}_2 = \frac{\sin(1 - \alpha)\theta}{\sin \theta} \mathbf{q}_0 + \frac{\sin \alpha \theta}{\sin \theta} \mathbf{q}_1, \quad (2.45)$$

where $\theta = \cos^{-1}(\mathbf{q}_0 \cdot \mathbf{q}_1)$ and the dot product is directly between the quaternion 4-vectors. All of these formulas give comparable results, although care should be taken when \mathbf{q}_0 and \mathbf{q}_1 are close together (which is why I prefer my formula, which uses an arctangent to establish the rotation angle).

Which rotation representation is better?

The choice of representation for 3D rotations depends partly on the application.

The axis/angle representation is minimal, and hence does not require any additional constraints on the parameters (no need to re-normalize after each update). If the angle is expressed in degrees,

it is easier to understand the pose (say 90° twist around x -axis), and also easier to express exact rotations. When the angle is in radians, the derivatives of \mathbf{R} w.r.t. ω can easily be computed (2.36).

Quaternions, on the other hand, are better if you want to keep track of a smoothly moving camera, since there are no discontinuities in the representation. It is also easier to interpolate between rotations and to chain rigid transformations together (Murray *et al.* 1994, Bregler and Malik 1998).

My usual preference is to use quaternions, but to update their estimates using an incremental rotation, as described in §5.2.2.

2.1.4 3D to 2D projections

Now that we know how to represent 2D and 3D geometric primitives and how to transform them spatially, we need to specify how 3D primitives get projected onto the image plane. We can do this using a linear 3D→2D projection matrix. The simplest model is orthography, which requires no division to get the final (inhomogeneous) result. The more commonly used model is perspective, since this more accurately models the behavior of real cameras.

Orthography and para-perspective

An orthographic projection simply drops the z component of the three-dimensional coordinate \mathbf{p} to obtain the 2D point \mathbf{x} . (In this section, I use \mathbf{p} to denote 3D points and \mathbf{x} to denote 2D points.) This can be written as

$$\mathbf{x} = [\mathbf{I}_{2 \times 2} | \mathbf{0}] \mathbf{p}. \quad (2.46)$$

If we are using homogeneous (projective) coordinates, we can write

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}, \quad (2.47)$$

i.e., we drop the z component but keep the w component. Orthography is an approximate model for long focal length (telephoto) lenses and/or objects whose depth is *shallow* relative to their distance to the camera (Sawhney and Hanson 1991). It is exact only for *telecentric* lenses (Baker and Nayar 1999, Baker and Nayar 2001).

In practice, world coordinates (which may measure dimensions in meters) need to be scaled to fit onto an image sensor (physically measured in millimeters, but ultimately measured in pixels). For this reason *scaled orthography* is actually more commonly used,

$$\mathbf{x} = [s\mathbf{I}_{2 \times 2} | \mathbf{0}] \mathbf{p}. \quad (2.48)$$

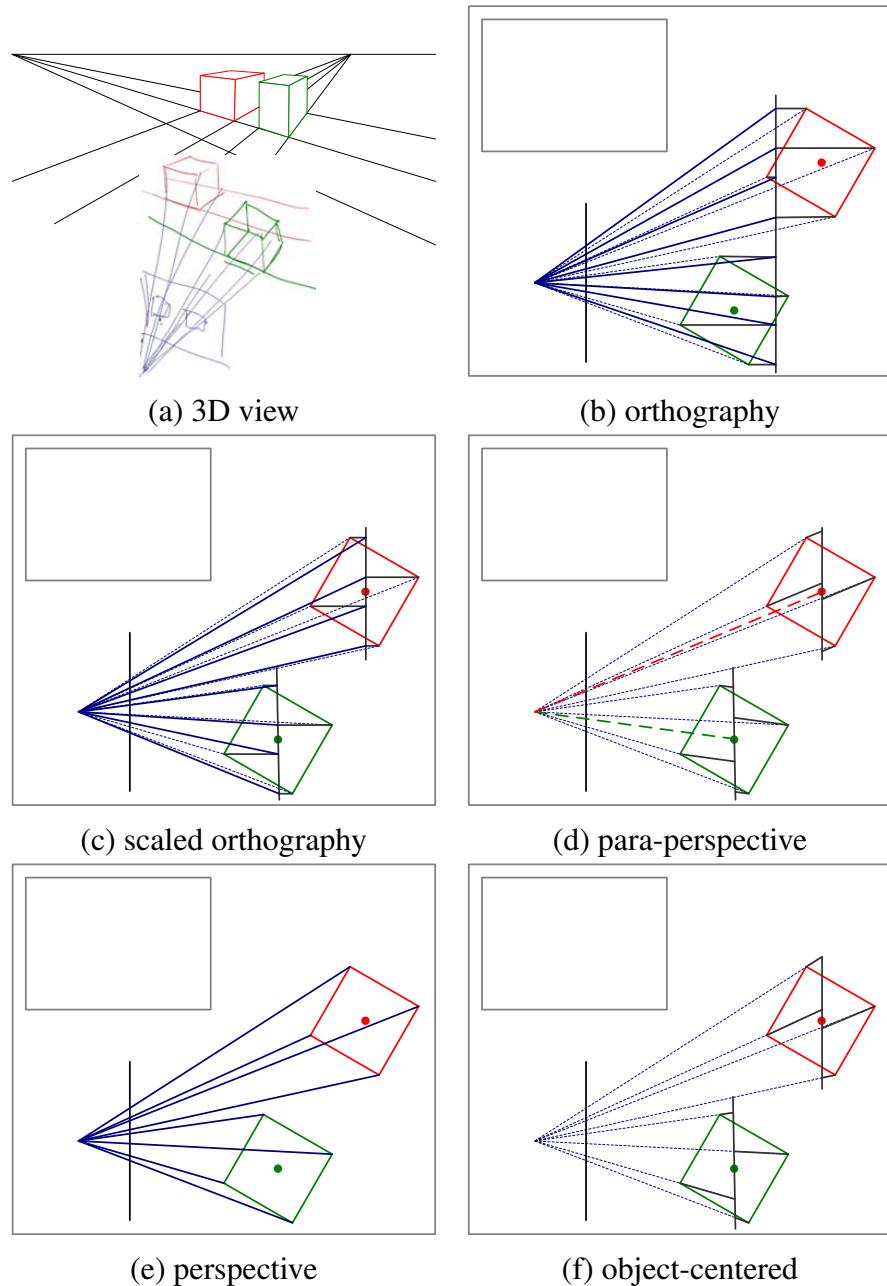


Figure 2.7: Commonly used projection models: (a) 3D view of world and camera; (b) orthography, (c) scaled orthography, (d) para-perspective, (e) perspective, (f) object-centered. Each diagram shows a top-down view of the projection model and a rendering of what the image appears like. Note how parallel lines on the ground plane and box sides remain parallel in the non-perspective projections.

[Note: Re-draw these figures by program once I have Exercise 2.3 implemented, including the insets, which show the 3D cubes projected under the given model.]

This model is equivalent to first projecting the world points onto a local fronto-parallel image plane, and then scaling this image using regular perspective projection. The scaling can either be the same for all parts of the scene, (Figure 2.7b), or it can be different for different objects that are being modeled independently (Figure 2.7c). More importantly, the scaling can vary from frame to frame when estimating *structure from motion*, which can better model the scale change that occurs as an object approaches the camera.

Scaled orthography is a popular model for reconstructing the 3D shape of objects far away from the camera, since it greatly simplifies certain computations. For example, *pose* (camera orientation) can be estimated using simple least squares §5.2.1. Under orthography, structure and motion can simultaneously be estimated using *factorization* (singular value decomposition) §6.3 (Tomasi and Kanade 1992).

A closely related projection model is *para-perspective* (Aloimonos 1990, Poelman and Kanade 1997). In this model, object points are again first projected onto a local reference parallel to the image plane. However, rather than being projected orthogonally to this plane, they are projected *parallel* to the light of sight to the object center (Figure 2.7c). This is followed by the usual projection onto the final image plane, which again amounts to a scaling. The combination of these two projections is therefore *affine* and can be written as

$$\tilde{\mathbf{x}} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}. \quad (2.49)$$

(Note how parallel lines in 3D remain parallel after projection in Figures 2.7b–d.) Para-perspective provides a more accurate projection model than scaled orthography, without incurring the added complexity of per-pixel perspective division, which invalidates traditional factorization methods (Poelman and Kanade 1997).

Perspective

The most commonly used projection in computer graphics and computer vision is true 3D *perspective* (Figure 2.7e.) Here, points are projected onto the image plane by dividing them by their z component. Using inhomogeneous coordinates, this can be written as

$$\mathbf{x} = \mathcal{P}_z(\mathbf{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}. \quad (2.50)$$

In homogeneous coordinates, the projection has a simple linear form,

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{p}}, \quad (2.51)$$

i.e., we drop the w component of \mathbf{p} . Thus, after projection, it is not possible to recover the *distance* of the 3D point from the image, which makes sense for a 2D imaging sensor.

A form often seen in computer graphics systems is a two-step projection that first projects 3D coordinates into *normalized device coordinates* in the range $(x, y, z) \in [-1, -1] \times [-1, 1] \times [0, 1]$, and then rescales these coordinates to integer pixel coordinates using a *viewport* transformation (Watt 1995, OpenGL-ARB 1997). The (initial) perspective projection is then represented using a 4×4 matrix

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-z_{\text{far}}}{z_{\text{far}} - z_{\text{near}}} & \frac{z_{\text{near}}z_{\text{far}}}{z_{\text{far}} - z_{\text{near}}} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{p}}, \quad (2.52)$$

where z_{near} and z_{far} and the near and far z *clipping planes*. (The first two rows are actually scaled by the focal length and the aspect ratio so that visible rays as mapped to $(x, y, z) \in [-1, -1]^2$. The reason for keeping the third row, rather than dropping it, is that visibility operations such as *z-buffering* require a depth for every graphical element that is being rendered.

If we set $z_{\text{near}} = 1$, $z_{\text{far}} \rightarrow \infty$, and switch the sign of the third row, the third element of the normalized screen vector becomes the inverse depth, i.e., the *disparity* (Okutomi and Kanade 1993). This can be quite convenient in many cases, since for cameras moving around in the outdoors, the inverse depth to the camera is often a better conditioned parameterization than direct 3D distance.

While a regular 2D image sensor has no way of measuring distance to a surface point, *range sensors* §11.4 or stereo matching algorithms §10 can compute such values. It is then convenient to be able to map from a sensor-based depth or disparity value d directly back to a 3D location using the inverse of a 4×4 matrix §2.1.4. We can do this if we represent perspective projection using a full-rank 4×4 matrix as in (2.63).

Camera intrinsics

[Note: This section is pretty long; a much simpler description exists in the stitching survey (Szeliski 2006a). Do I want to adopt something closer to that?]

Once we have projected a 3D point through an ideal pinhole using a projection matrix, we must still transform the resulting coordinates according to the pixel sensor spacing and the relative position of the sensor plane to the origin. Figure 2.8 shows an illustration of the geometry involved.

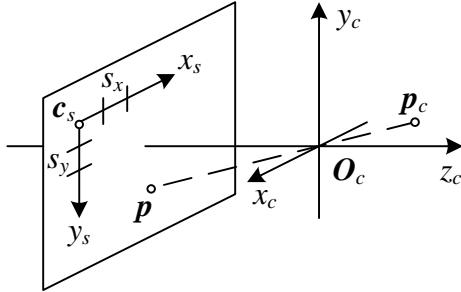


Figure 2.8: *Projection onto sensor plane and camera intrinsics.*

Image sensors return pixel values indexed by integer *pixel coordinates* (x_s, y_s) , often with the coordinates starting at the upper-left corner of the image and moving down and to the right. (This convention is not obeyed by all imaging libraries, but the adjustment for other coordinate systems is straightforward.) To map pixel centers to 3D coordinates, we first scale the (x_s, y_s) values by the pixel spacings (s_x, s_y) (sometimes expressed in microns for solid-state sensors), and then describe the orientation of the sensor array relative to the camera projection center O_c with an origin c_s and a 3D rotation R_s (Figure 2.8).

The combined 2D→3D projection can then be written as

$$\mathbf{p} = \left[\begin{array}{c|c} \mathbf{R}_s & \mathbf{c}_s \end{array} \right] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \mathbf{M}_s \mathbf{x}_s. \quad (2.53)$$

The first two columns of the 3×3 matrix \mathbf{M}_s are the 3D vectors corresponding to unit steps in the image pixel array along the x_s and y_s directions, while the third column is the 3D image array origin c_s .

The matrix \mathbf{M}_s is parameterized by eight unknowns: the rotation and translation $(\mathbf{R}_s, \mathbf{c}_s)$ describing the sensor orientation and the two scale factors (s_x, s_y) . (Note that we ignore here the possibility of *skew* between the two axes on the image plane, since solid-state manufacturing techniques render this negligible). In practice, unless we have accurate external knowledge of the sensor spacing and/or sensor orientation, there are only seven degrees of freedom, since the distance of the sensor from the origin cannot be teased apart from the sensor spacing, based on external image measurement alone.

However, estimating a camera model \mathbf{M}_s with the required seven degrees of freedom (i.e., where the first two columns are orthogonal after an appropriate re-scaling) is impractical, so most practitioners assume a general 3×3 homogeneous matrix form.

The relationship between the 3D pixel center \mathbf{p} and the 3D camera-centered point \mathbf{p}_c is given

by an unknown scaling s , $\mathbf{p} = s\mathbf{p}_c$. We can therefore write the complete projection between \mathbf{p}_c and a homogeneous version of the pixel address $\tilde{\mathbf{x}}_s$ as

$$\tilde{\mathbf{x}}_s = \alpha M_s^{-1} \mathbf{p}_c = \mathbf{K} \mathbf{p}_c. \quad (2.54)$$

The 3×3 matrix \mathbf{K} is called the *calibration matrix* and describes the camera *intrinsics* (as opposed to the camera's orientation in space, which are called the *extrinsics*).

From the above discussion, we see that in general, \mathbf{K} has seven degrees of freedom in theory, or 8 degrees of freedom (the full dimensionality of a 3×3 homogeneous matrix) in practice. Why, then, do most textbooks on 3D computer vision and multi-view geometry (Faugeras 1993, Hartley and Zisserman 2004, Faugeras and Luong 2001) treat \mathbf{K} as an upper-triangular matrix with five degrees of freedom?

While this is usually not made explicit in these books, it is because it is not possible to recover the full \mathbf{K} matrix based on external measurement alone. When calibrating a camera based on external 3D points or other measurements §5 (Tsai 1987), we end up estimating the intrinsic (\mathbf{K}) and extrinsic (\mathbf{R}, \mathbf{t}) camera parameters simultaneously using a series of measurements

$$\tilde{\mathbf{x}}_s = \mathbf{K} [\mathbf{R} | \mathbf{t}] \mathbf{p}_w, \quad (2.55)$$

where \mathbf{p}_w are known 3D world coordinates. Inspecting this equation, we see that we can post-multiply \mathbf{K} by \mathbf{R}_1 and pre-multiply $[\mathbf{R}|\mathbf{t}]$ by \mathbf{R}_1^T , and still end up with a valid calibration. Thus, it is impossible based on image measurements alone to know the true orientation of the sensor and the true camera intrinsics.

The choice of an upper-triangular form for \mathbf{K} seems to be conventional. Given a full 3×4 camera matrix $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$, we can compute an upper-triangular \mathbf{K} matrix using RQ factorization (Golub and Van Loan 1996). [Note: Check that an RQ algorithm does indeed exist, either in (Golub and Van Loan 1996) or in LAPack.] (Note the unfortunate clash of terminologies: In matrix algebra textbooks, \mathbf{R} represents an upper-triangular matrix, while in computer vision, \mathbf{R} is an orthogonal rotation.)

There are several ways to write the upper-triangular form of \mathbf{K} . One possibility is

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (2.56)$$

which uses independent *focal lengths* f_x and f_y for the sensor x and y dimensions. The entry s encodes any possible *skew* between the sensor axes due to the sensor not being mounted perpendicular to the optical axis, and (c_x, c_y) denotes the *optical center* expressed in pixel coordinates.

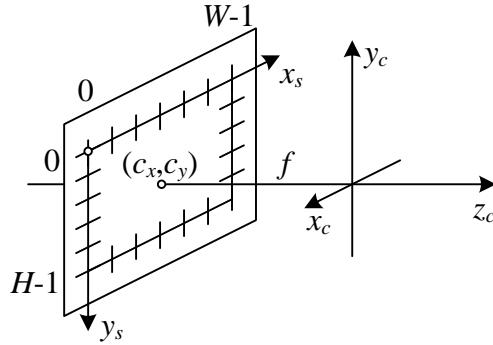


Figure 2.9: *Simplified camera intrinsics showing the focal length f and the optical center (c_x, c_y) . The image width and height are W and H .*

Another possibility is

$$\mathbf{K} = \begin{bmatrix} f & s & c_x \\ 0 & af & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (2.57)$$

where the *aspect ratio* a has been made explicit and a common focal length f is used.

In practice, for many applications, an even simpler form can be obtained by setting $a = 1$ and $s = 0$,

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.58)$$

Often, setting the origin at roughly the center of the image, e.g., $(c_x, c_y) = (W/2, H/2)$, where W and H are the image height and width, can result in a perfectly usable camera model with only a single unknown, i.e., the focal length f .

Figure 2.9 shows how these quantities can be visualized as part of a simplified imaging model. Note that now I have placed the image plane *in front* of the nodal point (projection center of the lens). The sense of the y axis has also been flipped to get a traditional *right-handed* coordinate system compatible with the way that most imaging libraries treat the vertical (row) coordinate. (Note that certain graphics libraries such as Direct3D use a left-handed coordinate system, which can lead to some confusion.) [*Note: Check what OpenGL does...*]

A note on focal lengths

The issue of how to express focal lengths f is one that often causes confusion in implementing computer vision algorithms and discussing their results. This is because the focal length depends on the units being used to measure pixels.

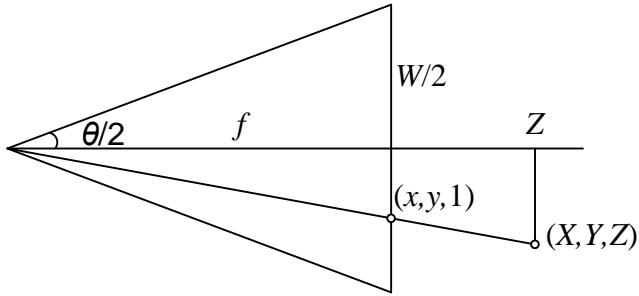


Figure 2.10: *Central projection, showing the relationship between the 3D and 2D coordinates p and x , as well as the relationship between the focal length f , image width W , and the field of view θ .*

If we number pixel coordinates using integer values, say $[0, W) \times [0, H)$, the focal length f and camera center (c_x, c_y) in (2.58) can be expressed as pixel values. How do these quantities relate to the more familiar focal lengths used by photographers?

Figure 2.10 illustrates the relationship between the focal length f , the sensor width W , and the field of view θ , which obey the formula

$$\tan \frac{\theta}{2} = \frac{W}{2f} \quad \text{or} \quad f = \frac{W}{2} \left[\tan \frac{\theta}{2} \right]^{-1}. \quad (2.59)$$

For conventional 35mm film cameras, $W = 35\text{mm}$, and hence f is also expressed in millimeters. Since we work with digital images, it is more convenient to express W in pixels so that the focal length f can be directly used in the calibration matrix K as in (2.58).

Another possibility is to scale the pixel coordinates so that they go from $[-1, 1]$ along the longer image dimension and $[-a^{-1}, a^{-1}]$ along the shorter axis, where $a \geq 1$ is the *image aspect ratio* (as opposed to the *sensor cell aspect ratio* introduced earlier). This can be accomplished using *modified normalized device coordinates*,

$$x'_s = (2x_s - W)/S \quad \text{and} \quad y'_s = (2y_s - H)/S, \quad \text{where} \quad S = \max(W, H). \quad (2.60)$$

This has the advantage that the focal length f and optical center (c_x, c_y) become independent of the image resolution, which can be useful when using multi-resolution image processing algorithms such as image pyramids §3.4.¹ The use of S instead of W also makes the focal length the same for landscape (horizontal) and portrait (vertical) pictures, as is the case in 35mm photography. (Note that in some computer graphics textbooks and systems, normalized device coordinates go

¹ To make the conversion truly accurate after a downsampling step in a pyramid, floating point values of W and H would have to be maintained since they can become non-integral if they are ever odd at a larger resolution in the pyramid.

from $[-1, 1] \times [-1, 1]$, which requires the use of two different focal lengths to describe the camera intrinsics (Watt 1995, OpenGL-ARB 1997).) Setting $S = W = 2$ in (2.59), we obtain the simpler (unit-less) relationship

$$f^{-1} = \tan \frac{\theta}{2}. \quad (2.61)$$

The conversion between the various focal length representations is straightforward, e.g., to go from a unitless f to one expressed in pixels, multiply by $W/2$, while to convert from an f expressed in pixels to the equivalent 35mm focal length, multiply by $35/W$.

Camera matrix

Now that we have shown how to parameterize the calibration matrix \mathbf{K} , we can put the camera intrinsics and extrinsics together to obtain a single 3×4 *camera matrix*

$$\mathbf{P} = \mathbf{K} \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \end{array} \right]. \quad (2.62)$$

It is sometimes preferable to use an invertible 4×4 matrix, which can be obtained by not dropping the last row in the \mathbf{P} matrix,

$$\tilde{\mathbf{P}} = \left[\begin{array}{cc} \mathbf{K} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{array} \right] \left[\begin{array}{cc} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{array} \right] = \tilde{\mathbf{K}}\mathbf{E}, \quad (2.63)$$

where \mathbf{E} is a 3D rigid-body (Euclidean) transformation and $\tilde{\mathbf{K}}$ is the full-rank calibration matrix. The 4×4 camera matrix $\tilde{\mathbf{P}}$ can be used to map directly from 3D world coordinates $\bar{\mathbf{p}}_w = (x_w, y_w, z_w, 1)$ to screen coordinates (plus disparity), $\mathbf{x}_s = (x_s, y_s, 1, d)$,

$$\mathbf{x}_s \sim \tilde{\mathbf{P}}\bar{\mathbf{p}}_w, \quad (2.64)$$

where \sim indicates equality up to scale. (Note that after multiplication by $\tilde{\mathbf{P}}$, the vector is divided by the *third* element of the vector to obtain the normalized form $\mathbf{x}_s = (x_s, y_s, 1, d)$.)

Plane plus parallax (projective depth)

In general, when using the 4×4 matrix $\tilde{\mathbf{P}}$, we have the freedom to remap the last row to whatever suits our purpose (rather than just being the “standard” interpretation of disparity as inverse depth). Let us re-write the last row of $\tilde{\mathbf{P}}$ as $\mathbf{p}_3 = s_3[\hat{\mathbf{n}}_0|c_0]$, where $\|\hat{\mathbf{n}}_0\| = 1$. We then have the equation

$$d = \frac{s_3}{z}(\hat{\mathbf{n}}_0 \cdot \mathbf{p}_w + c_0), \quad (2.65)$$

where $z = \mathbf{p}_2 \cdot \bar{\mathbf{p}}_w = \mathbf{r}_z \cdot (\mathbf{p}_w - \mathbf{c})$ is the distance of \mathbf{p}_w from the camera center c (2.25) along the optical axis z (Figure 2.11). Thus, we can interpret d as the *projective disparity* or *projective*

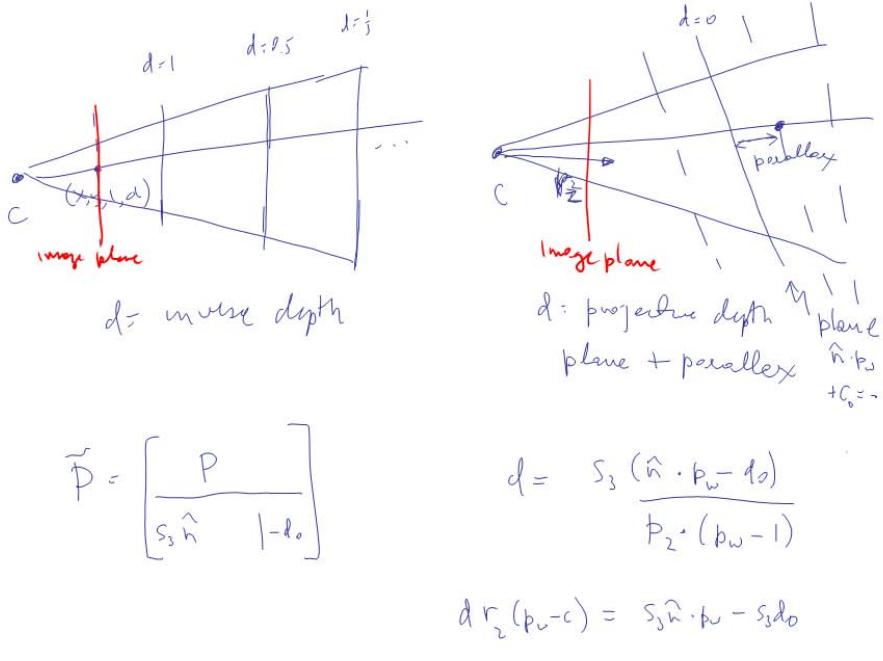


Figure 2.11: *Regular disparity d (inverse depth) and projective depth d (parallax from a reference plane).*

[Note: Regenerate these figures, using a program to get the spacing and orientation right on the projective depth figure.]

depth of a 3D scene point p_w from the *reference plane* $\hat{n} \cdot p_w + c_0 = 0$ (Szeliski and Coughlan 1997, Szeliski and Golland 1999, Shade *et al.* 1998, Baker *et al.* 1998). (The projective depth is also sometimes called *parallax* in reconstruction algorithm that use the term *plane plus parallax* (Kumar *et al.* 1994a, Sawhney 1994a, Sawhney 1994b).)

Another way to see this is to invert the \tilde{P} matrix so that we can map pixels plus disparity directly back to 3D points,

$$\tilde{p}_w = \tilde{P}^{-1} x_s. \quad (2.66)$$

In general, we can choose \tilde{P} to have whatever form is convenient, i.e., to sample space using an arbitrary projection. This can come in particularly handy when setting up multi-view stereo reconstruction algorithms, since it allows us to sweep a series of planes through space with a variable (projective) sampling that best matches the sensed image motions §10 (Collins 1996, Saito and Kanade 1999, Szeliski and Golland 1999).

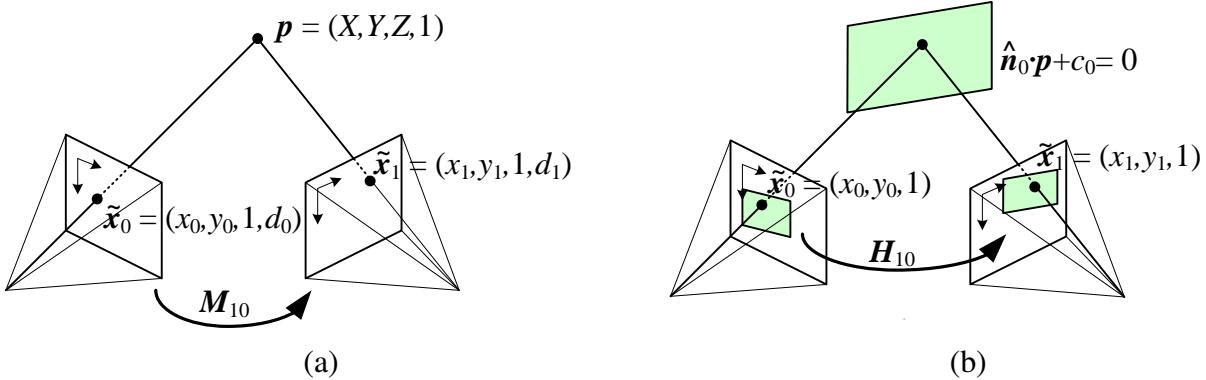


Figure 2.12: A point is projected into two images: (a) relationship between the 3D point coordinate $(X, Y, Z, 1)$ and the 2D projected point $(x, y, 1, d)$; (b) planar homography induced by points all lying on a common place $\hat{\mathbf{n}}_0 \cdot \mathbf{p} + c_0 = 0$.

Mapping from one camera to another

What happens when we take two images of a 3D scene from different camera positions and/or orientations (Figure 2.12a)? Using the full rank 4×4 camera matrix $\tilde{\mathbf{P}} = \tilde{\mathbf{K}}\mathbf{E}$ from (2.63), we can write the projection from world to screen coordinates as

$$\tilde{\mathbf{x}}_0 \sim \tilde{\mathbf{K}}_0 \mathbf{E}_0 \mathbf{p} = \tilde{\mathbf{P}}_0 \mathbf{p}. \quad (2.67)$$

Assuming that we know the z-buffer or disparity value d_0 for a pixel in one image, we can compute the 3D point location \mathbf{p} using

$$\mathbf{p} \sim \mathbf{E}_0^{-1} \tilde{\mathbf{K}}_0^{-1} \tilde{\mathbf{x}}_0 \quad (2.68)$$

and then project it into another image yielding

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{K}}_1 \mathbf{E}_1 \mathbf{p} = \tilde{\mathbf{K}}_1 \mathbf{E}_1 \mathbf{E}_0^{-1} \tilde{\mathbf{K}}_0^{-1} \tilde{\mathbf{x}}_0 = \tilde{\mathbf{P}}_1 \tilde{\mathbf{P}}_0^{-1} \tilde{\mathbf{x}}_0 = M_{10} \tilde{\mathbf{x}}_0. \quad (2.69)$$

Unfortunately, we do not usually have access to the depth coordinates of pixels in a regular photographic image. However, for a *planar scene*, as discussed above in (2.65), we can replace the last row of \mathbf{P}_0 in (2.63) with a general *plane equation*, $\hat{\mathbf{n}}_0 \cdot \mathbf{p} + c_0$ that maps points on the plane to $d_0 = 0$ values (Figure 2.12b). Thus, if we set $d_0 = 0$, we can ignore the last column of M_{10} in (2.69) and also its last row, since we do not care about the final z-buffer depth. The mapping equation (2.69) thus reduces to

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{H}}_{10} \tilde{\mathbf{x}}_0, \quad (2.70)$$

where $\tilde{\mathbf{H}}_{10}$ is a general 3×3 homography matrix and $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_0$ are now 2D homogeneous coordinates (i.e., 3-vectors) (Szeliski 1996). This justifies the use of the 8-parameter homography as a general alignment model for mosaics of planar scenes (Mann and Picard 1994, Szeliski 1996).

The other special case where we do not need to know depth to perform inter-camera mapping is when the camera is undergoing pure rotation (§8.1.3), i.e., when $t_0 = t_1$. In this case, we can write

$$\tilde{\mathbf{x}}_1 \sim \mathbf{K}_1 \mathbf{R}_1 \mathbf{R}_0^{-1} \mathbf{K}_0^{-1} \tilde{\mathbf{x}}_0 = \mathbf{K}_1 \mathbf{R}_{10} \mathbf{K}_0^{-1} \tilde{\mathbf{x}}_0, \quad (2.71)$$

which again can be represented with a 3×3 homography. If we assume that the calibration matrices have known aspect ratios and centers of projection (2.58), this homography can be parameterized by the rotation amount and the two unknown focal lengths. This particular formulation is commonly used in most image stitching applications (§8.1.3).

Object-centered projection

When working with long focal length lenses, it often becomes difficult to reliably estimate the focal length from image measurements alone. This is because the focal length and the distance to the object are highly correlated, and it becomes difficult to tease these two effects apart. For example, the change in scale of an object being viewed through a zoom telephoto lens can either be due to a zoom change or a motion towards the user. (This effect which was put to dramatic use in some of Alfred Hitchcock's film *Vertigo*, where the simultaneous change of zoom and camera motion produces a disquieting effect.)

This ambiguity becomes clearer if we write out the projection equation corresponding to the simple calibration matrix \mathbf{K} shown in (2.58),

$$\begin{aligned} x_s &= f \frac{\mathbf{r}_x \cdot \mathbf{p} + t_x}{\mathbf{r}_z \cdot \mathbf{p} + t_z} + c_x \\ y_s &= f \frac{\mathbf{r}_y \cdot \mathbf{p} + t_y}{\mathbf{r}_z \cdot \mathbf{p} + t_z} + c_y, \end{aligned} \quad (2.72)$$

where \mathbf{r}_x , \mathbf{r}_y , and \mathbf{r}_z are the three rows of \mathbf{R} . If the distance to the object center $t_z \gg \|\mathbf{p}\|$ (the size of the object), the denominator is approximately t_z and the overall scale of the projected object depends on the ratio of f to t_z . It therefore becomes difficult to disentangle these two quantities.

To see this more clearly, let $\eta_z = t_z^{-1}$ and $s = \eta_z f$. We can then re-write the above equations as

$$\begin{aligned} x_s &= s \frac{\mathbf{r}_x \cdot \mathbf{p} + t_x}{1 + \eta_z \mathbf{r}_z \cdot \mathbf{p}} + c_x \\ y_s &= s \frac{\mathbf{r}_y \cdot \mathbf{p} + t_y}{1 + \eta_z \mathbf{r}_z \cdot \mathbf{p}} + c_y \end{aligned} \quad (2.73)$$

(Szeliski and Kang 1994, Pighin *et al.* 1998). The scale of the projection s can be reliably estimated if we are looking at a known object (i.e., the 3D coordinates \mathbf{p} are known). The inverse distance η_z is now mostly decoupled from the estimates of s and can be estimated from the amount of



Figure 2.13: Examples of radial lens distortion: (a) barrel, (b) pincushion, and (c) fisheye. The fisheye image spans almost a complete 180° from side-to-side.

foreshortening as the object rotates. Furthermore, as the lens becomes longer, i.e., the projection model becomes orthographic, there is no need to replace a perspective imaging model with an orthographic one, since the same equation can be used, with $\eta_z \rightarrow 0$ (as opposed to f and t_z both going to infinity). This allows us to form a natural link between orthographic reconstruction techniques such as factorization and their projective/perspective counterparts §6.3.

2.1.5 Lens distortions

The above imaging models all assume that cameras obey a *linear* projection model where straight lines in the world result in straight lines in the image. (This follows as a natural consequence of linear matrix operations being applied to homogeneous coordinates.) Unfortunately, many wide angle lenses have noticeable *radial distortion*, which manifests itself as a visible curvature in the projection of straight lines. (See §2.2.3 for a more detailed discussion of lens optics, including chromatic aberration.) Unless this distortion is taken into account, it becomes impossible to create highly accurate photorealistic reconstructions. For example, image mosaics constructed without taking radial distortion into account will often exhibit blurring due to the mis-registration of corresponding features before pixel blending §8.

Fortunately, compensating for radial distortion is not that difficult in practice. For most lenses, a simple quartic model of distortion can produce good results. Let (x_c, y_c) be the pixel coordinates obtained *after* perspective division but *before* scaling by focal length f and shifting by the optical center (c_x, c_y) , i.e.,

$$x_c = \frac{\mathbf{r}_x \cdot \mathbf{p} + t_x}{\mathbf{r}_z \cdot \mathbf{p} + t_z}$$

$$y_c = \frac{\mathbf{r}_y \cdot \mathbf{p} + t_y}{\mathbf{r}_z \cdot \mathbf{p} + t_z}. \quad (2.74)$$

The radial distortion model says that coordinates in the observed images are displaced away (*barrel* distortion) or towards (*pincushion* distortion) the image center by an amount proportional to their radial distance (Figure 2.13a–b). The simplest radial distortion models use low-order polynomials, e.g.,

$$\begin{aligned} x'_c &= x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \\ y'_c &= y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4), \end{aligned} \quad (2.75)$$

where $r_c^2 = x_c^2 + y_c^2$ and κ_1 and κ_2 are called the *radial distortion parameters* (Slama 1980). After the radial distortion step, the final pixel coordinates can be computed using

$$\begin{aligned} x_s &= f x'_c + c_x \\ y_s &= f y'_c + c_y. \end{aligned} \quad (2.76)$$

[Note: Sometimes the relationship between x_c and x'_c is expressed the other way around. This is convenient if we map image pixels into (warped) rays by dividing through by f . We can then undistort the rays and have true 3D rays in space. I will need to work this out after I write the calibration section.] A variety of techniques can be used to estimate the radial distortion parameters for a given lens, as discussed in §5.3.5.

Sometimes the above simplified model does not model the true distortions produced by complex lenses accurately enough (especially at very wide angles). A more complete analytic model also includes *tangential distortions* and *decentering distortions* (Slama 1980), but these will not be covered in this book.

Fisheye lenses require a different model than traditional polynomial models of radial distortion (Figure 2.13c). Instead, fisheye lenses behave, to a first approximation, as *equi-distance* projectors of angles away from the optical axis (Xiong and Turkowski 1997), which is the same as the *polar projection* described by equations (8.28–8.30) in §8.1. Xiong and Turkowski (1997) describe how this model can be extended with the addition of an extra quadratic correction in ϕ , and how the unknown parameters (center of projection, scaling factor s , etc.) can be estimated from a set of overlapping fisheye images using a direct (intensity-based) non-linear minimization algorithm.

For even larger, less regular distortions, a parametric distortion model using splines may be necessary (Goshtasby 1989). If the lens does not have a single center of projection, it may become necessary to model the 3D *line* (as opposed to *direction*) corresponding to each pixel separately (Gremban *et al.* 1988, Champlouboux *et al.* 1992a, Grossberg and Nayar 2001, Sturm and Rama-lingam 2004, Tardif *et al.* 2006). Some of these techniques are described in more detail in §5.3.5, which discusses how to calibrate lens distortions.

[Note: Drop the following paragraph?] There is one subtle issue associated with the simple radial distortion model that is often glossed over. We have introduced a non-linearity between the perspective projection and final sensor array projection steps. Therefore, we cannot in general post-multiply an arbitrary 3×3 matrix \mathbf{K} with a rotation to put it into upper-triangular form and absorb this into the global rotation. However, this situation is not as bad as it may at first appear. For many applications, keeping the simplified diagonal form of (2.58) is still an adequate model. Furthermore, if we correct radial and other distortions to an accuracy where straight lines are preserved, we have essentially converted the sensor back into a linear imager, and the previous decomposition still applies.

2.2 Photometric image formation

In modeling the image formation process, I have described how 3D geometric features in the world are projected into 2D features in an image. However, images are not composed of 2D features. Instead, they are made up of discrete color or intensity values. Where do these values come from? How do they relate to the lighting in the environment, surface properties and geometry, camera optics, and sensor properties (Figure 2.14)? In this section, I develop a set of models to describe these interactions and formulate a generative process of image formation. A more detailed treatment of these topics can be found in (Glassner 1995) and other textbooks on computer graphics and image synthesis (Foley *et al.* 1995, Watt 1995, Cohen and Wallace 1993, Sillion and Puech 1994).

2.2.1 Lighting

Images cannot exist without light. To produce an image, the scene must be illuminated with one or more light sources. (Certain modalities such as fluorescent microscopy and X-ray tomography do not fit this model, but I do not deal with them in this book.) Light sources can generally be divided into point and area light sources.

A point light source originates at a single location in space (e.g., a small light bulb), potentially at infinity (e.g., the sun). (Note that for some applications such as modeling soft shadows (*penumbras*), the sun may have to be treated as an areal source.) In addition to its location, a point light source has an intensity and a color spectrum, i.e., a distribution over wavelengths $L(\lambda)$. The intensity of a light source falls off with the square of the distance between the source and the object being lit, because the same light is being spread over a larger (spherical) area. A light source may also have a directional falloff (dependence), but we ignore this in our simplified model.

Area light sources are more complicated. A simple area light source such as a fluorescent ceiling light fixture with a diffuser can be modeled as a finite rectangular area emitting light equally

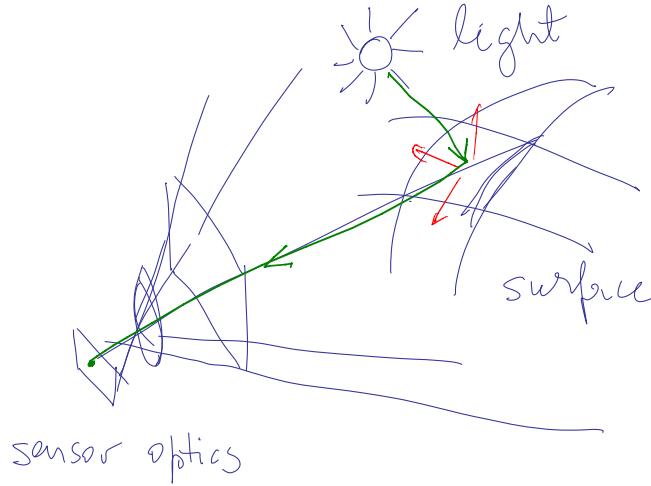


Figure 2.14: *A simplified model of photometric image formation. Light is emitted by one or more light sources, and is then reflected from an object’s surface. A portion of this light is directed towards the camera. This simplified model ignores multiple reflections, which often occur in real-world scenes.*

in all directions (Cohen and Wallace 1993, Sillion and Puech 1994, Glassner 1995). When the distribution is strongly directional, a four-dimensional lightfield can be used instead (Ashdown 1993).

A more complex light distribution that approximates, say, the incident illumination on an object sitting in an outdoor courtyard, can often be represented using an *environment map* (Greene 1986) (originally called a *reflection map* in (Blinn and Newell 1976)). This representation maps incident light directions \hat{v} to color values (or wavelengths, λ),

$$L(\hat{v}; \lambda), \quad (2.77)$$

and is equivalent to assuming that all light sources are at infinity. Environment maps can be represented as a collection of cubical faces (Greene 1986), as a single longitude-latitude map (Blinn and Newell 1976), or as the image of a reflecting sphere (Watt 1995). A convenient way to get a rough model of a real-world environment map is to take an image of a reflective mirrored sphere and to then unwrap this image onto the desired environment map (Debevec 1998). Watt (1995) has a nice discussion of environment mapping, including the formulas needed to map directions to pixels for the three most commonly used representations.

2.2.2 Reflectance and shading

When light hits an object’s surface, it is scattered and reflected (Figure 2.15a). Many different models have been developed to describe this interaction. In this section, I first describe the most

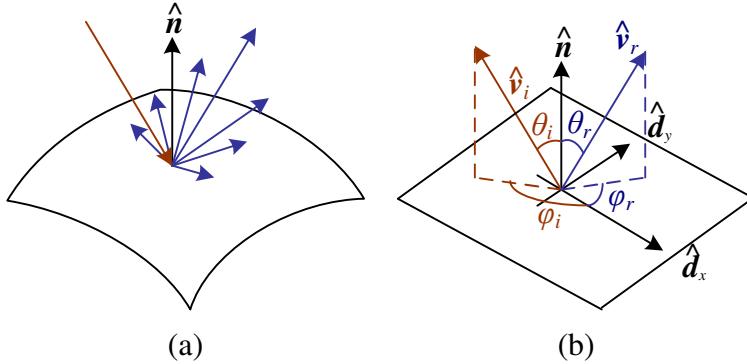


Figure 2.15: (a) Light scattering when hitting a surface. (b) The bidirectional reflectance distribution function (BRDF) $f(\theta_i, \phi_i, \theta_r, \phi_r)$ is parameterized by the angles the incident \hat{v}_i and reflected \hat{v}_r light ray directions make with the local surface coordinate frame ($\hat{d}_x, \hat{d}_y, \hat{n}$).

general form, namely the BRDF, and then look at some more specialized models, including the diffuse, specular, and Phong shading models. I also discuss how these models can be used to compute the *global illumination* corresponding to a scene.

The Bidirectional Reflectance Distribution Function (BRDF)

The most general model of light scattering is the *bidirectional reflectance distribution function* (BRDF). Relative to some local coordinate frame on the surface, the BRDF is a four-dimensional function that describes how much of each wavelength arriving at an *incident* direction \hat{v}_i is emitted in a *reflected* direction \hat{v}_r (Figure 2.15b). The function can be written in terms of the angles of the incident and reflected directions relative to the surface frame as

$$f_r(\theta_i, \phi_i, \theta_r, \phi_r; \lambda). \quad (2.78)$$

The BRDF is *reciprocal*, i.e., because of the physics of light transport, you can interchange the roles of \hat{v}_i and \hat{v}_r and still get the same answer (this is sometimes called *Helmholtz reciprocity*).

Most surfaces are *isotropic*, i.e., there are no preferred directions on the surface as far as light transport is concerned. (The exceptions are *anisotropic* surfaces such as brushed (scratched) aluminum, where the reflectance depends on the light orientation relative to the direction of the scratches.) For an isotropic material, we can simplify the BRDF to

$$f_r(\theta_i, \theta_r, |\phi_r - \phi_i|; \lambda) \text{ or } f_r(\hat{v}_i, \hat{v}_r, \hat{n}; \lambda), \quad (2.79)$$

since the quantities θ_i, θ_r and $\phi_r - \phi_i$ can be computed from the directions \hat{v}_i, \hat{v}_r , and \hat{n} .

To calculate the amount of light exiting a surface point p in a direction \hat{v}_r under a given lighting condition, we integrate the product of the incoming light $L_i(\hat{v}_i; \lambda)$ with the BRDF (some

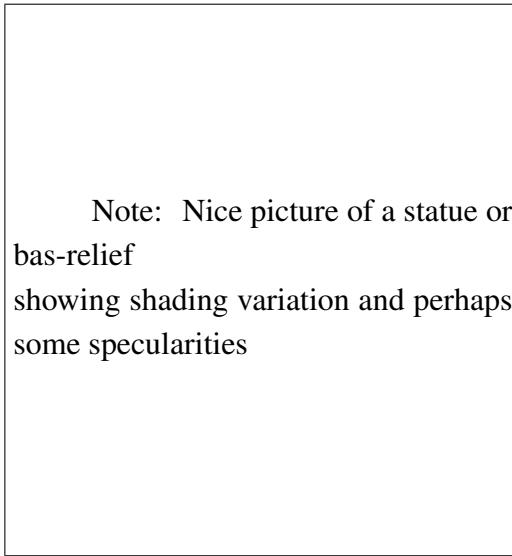


Figure 2.16: This close-up of a statue shows both diffuse (smooth shading) and specular (shiny highlight) reflection.

authors call this step a *convolution*). Taking into account the *foreshortening* factor $\cos^+ \theta_i$ (which is explained in (2.84)), we obtain

$$L_r(\hat{\mathbf{v}}_r; \lambda) = \int L_i(\hat{\mathbf{v}}_i; \lambda) f_r(\hat{\mathbf{v}}_i, \hat{\mathbf{v}}_r, \hat{\mathbf{n}}; \lambda) \cos^+ \theta_i d\hat{\mathbf{v}}_i. \quad (2.80)$$

If the light sources are discrete (a finite number of point light sources), we can replace the integral with a summation,

$$L_r(\hat{\mathbf{v}}_r; \lambda) = \sum_i L_i(\lambda) f_r(\hat{\mathbf{v}}_i, \hat{\mathbf{v}}_r, \hat{\mathbf{n}}; \lambda) \cos^+ \theta_i. \quad (2.81)$$

BRDFs for a given surface can be obtained through physical modeling (Torrance and Sparrow 1967, Cook and Torrance 1982, Glassner 1995), heuristic modeling (Phong 1975), or through empirical observation, e.g., (Ward 1992, Westin *et al.* 1992, Dana *et al.* 1999).² Typical BRDFs can often be split into their *diffuse* and *specular* components, as described below.

Diffuse reflection

The diffuse component (also known as *Lambertian* or *matte* reflection) scatters light uniformly in all directions and is the phenomenon we most normally associate with *shading*, e.g., the smooth (non-shiny) variation of intensity with surface normal seen when observing a statue (Figure 2.16). Diffuse reflection also often imparts a strong *body color* to the light since it is caused by selective absorption and re-emission of light inside the object's material (Shafer 1985, Glassner 1995).

² See <http://www1.cs.columbia.edu/CAVE/software/curet/> for a database of some empirically sampled BRDFs.

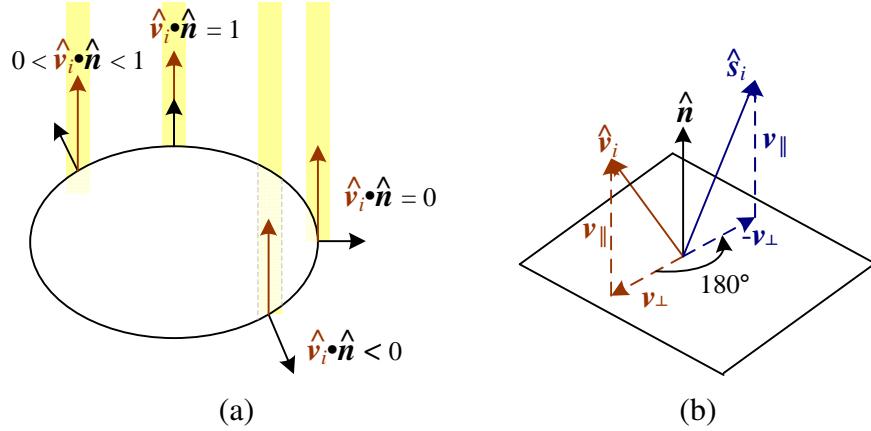


Figure 2.17: (a) The diminution of returned light caused by foreshortening depends on $\hat{v}_i \cdot \hat{n}$, the cosine of the angle between the incident light direction \hat{v}_i and the surface normal \hat{n} . (b) Mirror (specular) reflection: The incident light ray direction \hat{v}_i is reflected onto the specular direction \hat{s}_i around the surface normal \hat{n} .

While light is scattered uniformly in all directions, i.e., the BRDF is constant,

$$f_d(\hat{v}_i, \hat{v}_r, \hat{n}; \lambda) = f_d(\lambda), \quad (2.82)$$

the amount of light depends on the angle between the incident light direction and the surface normal θ_i . This is because the surface area exposed to a given amount of light becomes larger at oblique angles, becoming completely self-shadowed as the outgoing surface normal points away from the light (Figure 2.17a). (Think about how you orient yourself towards the sun or fireplace to get maximum warmth, or how a flashlight projected obliquely against a wall is less bright than one pointing directly at it.) The *shading equation* for diffuse reflection can thus be written as

$$L_d(\hat{v}_r; \lambda) = \sum_i L_i(\lambda) f_d(\lambda) \cos^+ \theta_i = \sum_i L_i(\lambda) f_d(\lambda) [\hat{v}_i \cdot \hat{n}]^+, \quad (2.83)$$

where

$$\cos^+ \theta_i = \max(0, \cos \theta_i) \text{ and } [\hat{v}_i \cdot \hat{n}]^+ = \max(0, \hat{v}_i \cdot \hat{n}). \quad (2.84)$$

Specular reflection

The second major component of a typical BRDF is *specular* (a.k.a. gloss or highlight) reflection, which depends strongly on the direction of the outgoing light. Consider light reflecting off a mirrored surface (Figure 2.17b). Incident light rays are reflected in a direction that is rotated by 180° around the surface normal \hat{n} . Using the same notation as in (2.29–2.30), we can compute the *specular reflection* direction \hat{s}_i as

$$\hat{s}_i = \mathbf{v}_{\parallel} - \mathbf{v}_{\perp} = (2\hat{n}\hat{n}^T - \mathbf{I})\mathbf{v}_i. \quad (2.85)$$

The amount of light reflected in a given direction $\hat{\mathbf{v}}_r$ thus depends on the angle $\theta_s = \cos^{-1}(\hat{\mathbf{v}}_r \cdot \hat{\mathbf{s}}_i)$ between the view direction $\hat{\mathbf{v}}_r$ and the specular direction $\hat{\mathbf{s}}_i$. For example, the [Phong \(1975\)](#) model uses a power of the cosine of the angle,

$$f_s(\theta_s; \lambda) = k_s(\lambda) \cos^{k_e} \theta_s, \quad (2.86)$$

while the [Torrance and Sparrow \(1967\)](#) micro-facet model uses a Gaussian,

$$f_s(\theta_s; \lambda) = k_s(\lambda) \exp(-c_s^2 \theta_s^2). \quad (2.87)$$

Larger exponents k_e (or inverse Gaussian widths c_s) correspond to more specular surfaces with distinct highlights, while smaller exponents better model materials with softer gloss.

Phong shading

[Phong \(1975\)](#) combined the diffuse and specular components of reflection together with another term, which he called the *ambient illumination*. This term accounts for the fact that objects are generally not only illuminated by point light sources, but also by a general diffuse illumination corresponding to inter-reflection (e.g., the walls in a room) or distant sources such as the blue sky. In the Phong model, the ambient term does not depend on surface orientation, but depends on the color of both the ambient illumination $L_a(\lambda)$ and the object color $k_a(\lambda)$,

$$f_a(\lambda) = k_a(\lambda) L_a(\lambda). \quad (2.88)$$

Putting all of these terms together, we arrive at the *Phong shading* model,

$$L_r(\hat{\mathbf{v}}_r; \lambda) = k_a(\lambda) L_a(\lambda) + k_d(\lambda) \sum_i L_i(\lambda) [\hat{\mathbf{v}}_i \cdot \hat{\mathbf{n}}]^+ + k_s(\lambda) \sum_i L_i(\lambda) (\hat{\mathbf{v}}_r \cdot \hat{\mathbf{s}}_i)^{k_e}. \quad (2.89)$$

[Note: Show some cross sections through a BRDF function a fixed incident direction with different amounts of specularity.]

Typically, the the ambient and diffuse reflection color distributions $k_a(\lambda)$ and $k_d(\lambda)$ are the same, since they are both due to sub-surface scattering (body reflection) inside the surface material ([Shafer 1985](#)). The specular reflection distribution $k_s(\lambda)$ is often uniform (white), since it is caused by interface reflections that do not change the light color. (The exception to this are *metallic* materials such as copper, as opposed to the more common *dielectric* materials such as plastics.)

The ambient illumination $L_a(\lambda)$ often has a different color cast than the direct light sources $L_i(\lambda)$, e.g., it may be blue for a sunny outdoors scene, or yellow for an interior lit with candlelight or incandescents. (The presence of ambient sky illumination in shadowed areas is what often causes shadows to appear bluer than the corresponding lit portions of a scene). [Note: Add a photo showing this?] Note also that the diffuse component of the Phong model (or in general, of any

shading model) depends on the angle of the *incoming* light source $\hat{\mathbf{v}}_i$, while the specular component depends on the relative angle between the viewer \mathbf{v}_r and the specular reflection direction $\hat{\mathbf{s}}_i$ (which itself depends on the incoming light direction $\hat{\mathbf{v}}_i$ and the surface normal $\hat{\mathbf{n}}$).

The Phong shading model has been superseded in terms of physical accuracy by a number of more recently developed models in computer graphics, including the model developed by Cook and Torrance (1982) based on the original microfacet model of Torrance and Sparrow (1967). Until recently, most computer graphics hardware only implemented the Phong model, but the recent advent of programmable pixel shaders makes the use of more complex models feasible.

The Torrance and Sparrow model of reflection also forms the basis of Shafer's (1985) *di-chromatic reflection model*, which states that the apparent color of a uniform material lit from a single source depends on the sum of two terms,

$$L_r(\hat{\mathbf{v}}_r; \lambda) = L_i(\hat{\mathbf{v}}_r, \hat{\mathbf{v}}_i, \hat{\mathbf{n}}; \lambda) + L_b(\hat{\mathbf{v}}_r, \hat{\mathbf{v}}_i, \hat{\mathbf{n}}; \lambda) \quad (2.90)$$

$$= c_i(\lambda)m_i(\hat{\mathbf{v}}_r, \hat{\mathbf{v}}_i, \hat{\mathbf{n}}) + c_b(\lambda)m_b(\hat{\mathbf{v}}_r, \hat{\mathbf{v}}_i, \hat{\mathbf{n}}), \quad (2.91)$$

i.e., the radiance of the light reflected at the *interface*, L_i , and the radiance reflected at the *surface body*, L_b . Each of these, in turn, is a simple product between a relative power spectrum $c(\lambda)$, which depends only on wavelength, and a magnitude $m(\hat{\mathbf{v}}_r, \hat{\mathbf{v}}_i, \hat{\mathbf{n}})$, which depends only on geometry. (This model can easily be derived from a generalized version of Phong's model by assuming a single light source, no ambient illumination, and re-arranging terms.) The di-chromatic model has been successfully used in computer vision to segment specular colored objects with large variations in shading (Klinker 1993) and more recently has inspired local two-color models for applications such as Bayer pattern de-mosaicing (Bennett *et al.* 2006).

Global illumination (ray tracing and radiosity)

The simple shading model presented thus far assumes that light rays leave the light source(s), bounce off of surfaces visible to the camera, thereby changing in intensity and/or color, and arrive at the camera. In reality, light sources can be shadowed by occluders, and rays can bounce multiple times around a scene while making their trip for a light source to the camera.

Two different methods have traditionally been used to model such effects. If the scene is mostly specular (the classic example being scenes made of glass objects and mirrored or highly polished balls), as shown in Figure 2.18a, the preferred approach is *ray tracing* or *path tracing* (Glassner 1995, Akenine-Möller and Haines 2002, Shirley 2005), which follows individual rays from the camera across multiple bounces towards the light source(s) (or vice versa). If the scene is composed mostly of uniform albedo simple geometry illuminators and surfaces, *radiosity* (*global illumination*) techniques are preferred (Cohen and Wallace 1993, Sillion and Puech 1994, Glassner 1995). Combinations of the two techniques have also been developed (Wallace *et al.* 1987), as

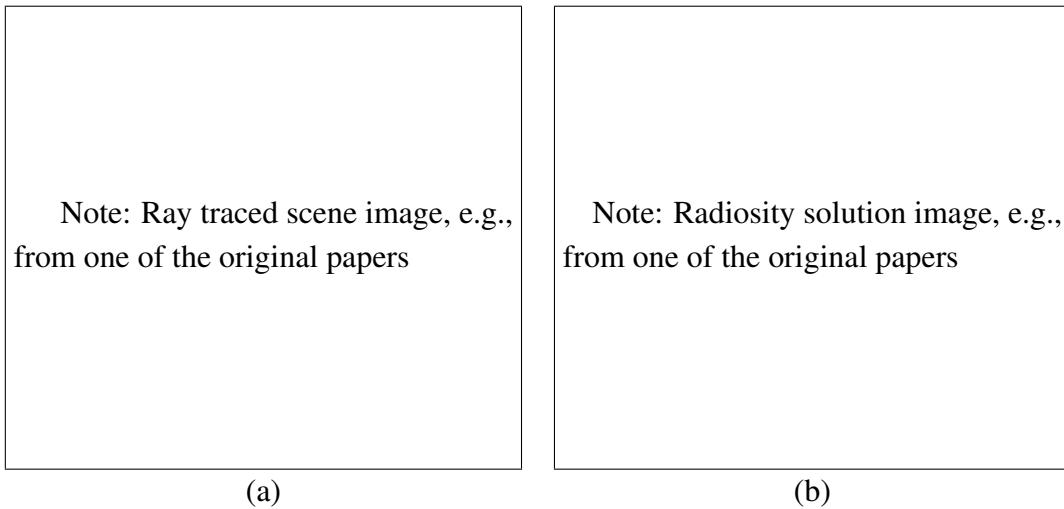


Figure 2.18: *Global illumination effects: (a) ray-traced image; (b) radiosity solution.*

well as more general *light transport* techniques for simulating effects such as the *caustics* cast by rippling water.

The basic ray tracing algorithm associates a light ray with each pixel in the camera image and finds its intersection with the nearest surface. A *primary* contribution can then be computed using the simple shading equations presented previously, e.g., (2.89), for all light sources that are visible for that surface element. (An alternative technique for computing which surfaces are illuminated by a light source is to compute a *shadow map* (or *shadow buffer*), i.e., a rendering of the scene from the light source's perspective, and to then compare the depth of pixels being rendered with the map (Williams 1978, Akenine-Möller and Haines 2002).) Additional *secondary* rays can then be cast along the specular direction towards other objects in the scene, keeping track of any attenuation or color change that the specular reflection induces, as shown in Figure 2.19a.

Radiosity works by associating lightness values with rectangular surface areas in the scene (including area light sources). The amount of light interchanged between any two (mutually visible) areas in the scene can be captured as a *form factor* (Figure 2.19b), which depends on their relative orientation and surface reflectance properties, as well as the $1/r^2$ fall-off as light gets distributed over a larger effective sphere the further away it is (Cohen and Wallace 1993, Sillion and Puech 1994, Glassner 1995). A large linear system can then be set up to solve for the final lightness of each area patch, using the light sources as the forcing function (right hand side). Once the system has been solved, the scene can be rendered from any desired point of view. Under certain circumstances, it is possible to recover the global illumination in a scene from photographs using computer vision techniques (Yu *et al.* 1999).

The basic radiosity algorithm does not take into account certain *near field* effects, such as the darkening inside corners and scratches, or the limited ambient illumination caused by partial

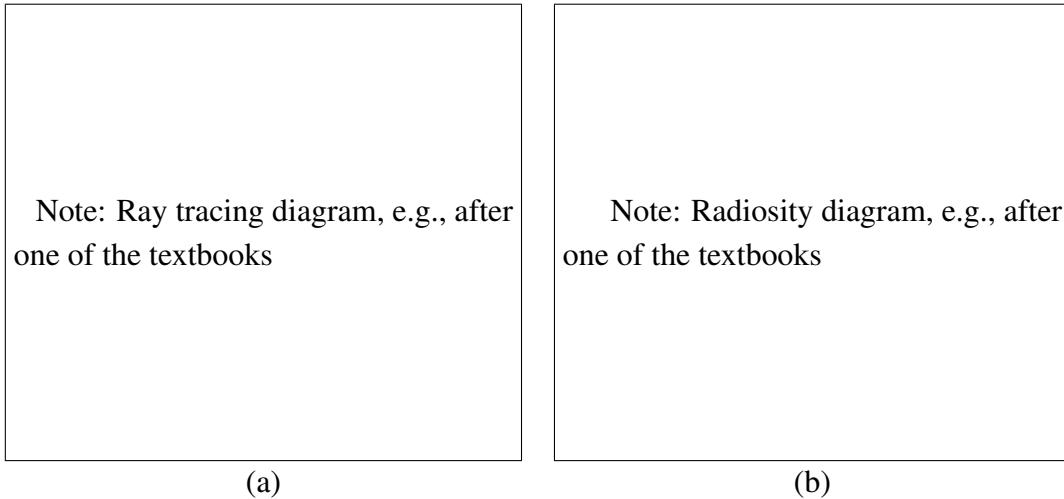


Figure 2.19: *Global illumination techniques*: (a) *ray tracing, with primary and secondary rays*; (b) *radiosity, with interacting areal form factors*.

shadowing from other surfaces. Such effects have been exploited in a number of computer vision algorithms (Nayar *et al.* 1991, Langer and Zucker 1994).

While all of these global illumination effects can have a strong effect on the appearance of a scene, and hence its 3D interpretation, they will not be covered in more detail in this book. (But see §11.8.1 for a discussion of recovering BRDFs from real scenes and objects.)

2.2.3 Optics

Once the light from the scene reaches the camera, it must still pass through the lens before reaching the sensor (analog film or digital silicon). For many applications, it suffices to treat the lens as an ideal pinhole that simply projects all rays through a common center of projection (Figures 2.8 and 2.9).

However, if we want to deal with issues such as focus, exposure, vignetting, and aberration, we need to develop a more sophisticated model, which is where the study of *optics* comes in (Möller 1988, Hecht 2001, Ray 2002).

Figure 2.20 shows a diagram of the most basic lens model, i.e., the *thin lens* composed of a single piece of glass with very low, equal curvature on both sides. According to the *lens law* (which can be derived using simple geometric arguments on light ray refraction), the relationship between the distance to an object z_o and the distance behind the lens at which a focused image is formed z_i can be expressed as

$$\frac{1}{z_o} + \frac{1}{z_i} = \frac{1}{f}, \quad (2.92)$$

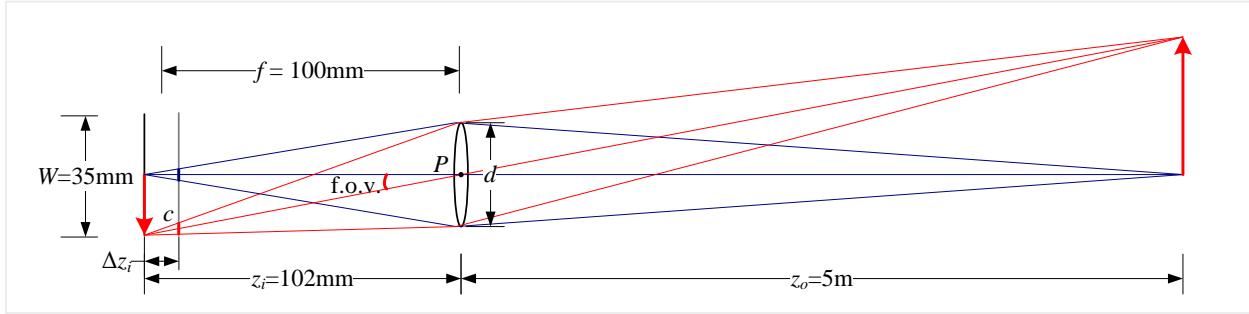


Figure 2.20: A thin lens of focal length f focuses the light from a plane a distance z_o in front of the lens at a distance z_i behind the lens, where $\frac{1}{z_o} + \frac{1}{z_i} = \frac{1}{f}$. If the focal plane is moved forward (vertical gray line), the images are no longer in focus, and the circle of confusion c (small thick red line segment) depends on the distance of the image plane motion Δz_i relative to the lens aperture diameter d . The field of view (f.o.v.) depends on the ratio between the sensor width W and the focal length f (or more precisely, the focusing distance z_i , which is usually quite close to f).

where f is called the *focal length* of the lens. If we let $z_o \rightarrow \infty$, i.e., we adjust the lens (move the image plane) so that objects at infinity are in focus, we get $z_i = f$, which is why we can think of a lens of focal length f as being equivalent (to a first approximation) to a pinhole a distance f from the focal plane (Figure 2.10), whose field of view is given by (2.59).

If the focal plane is moved away from its proper in-focus setting of z_i (i.e., by twisting the focus ring on the lens), objects at z_o are no longer in focus, as shown by the gray plane in Figure 2.20. The amount of mis-focus is measured by the *circle of confusion* c (shown as small blue and red lines on the gray plane).³ The equation for the circle of confusion can be derived using similar triangles, and depends on the distance of travel in the focal plane Δz_i relative to the original focus distance z_i and the diameter of the aperture d (Exercise 2.4).

The allowable depth variation in the scene that limits the circle of confusion to an acceptable number is commonly called the *depth of field* and is a function of both the focus distance and the aperture, as shown diagrammatically on a lot of lens markings (Figure 2.21). Since this depth of field depends on the aperture diameter d , we also have to know how this varies with the commonly displayed *f-number*, which is usually denoted as $f/\#$ or N and is defined as

$$f/\# = N = \frac{f}{d}, \quad (2.93)$$

where the focal length f and the aperture diameter d are measured in the same unit (say millimeters).

³ If the aperture is not completely circular, e.g., if it is caused by a hexagonal diaphragm, it is sometimes possible to see this effect in the actual blur function (Levin *et al.* 2007, Joshi *et al.* 2008) or in the “glints” that are seen when shooting into the sun.

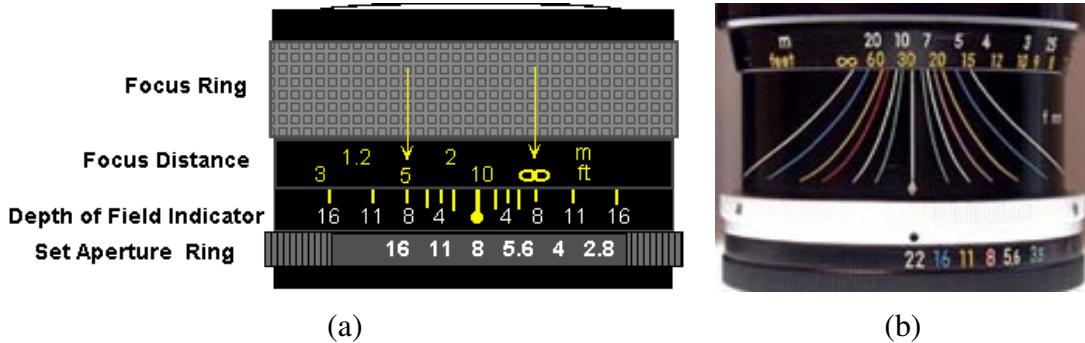


Figure 2.21: Regular and zoom lens width depth of field indicators

[Note: Replace these Internet photos with ones I take myself]

The usual way to write the f-number is to replace the # in $f/#$ with the actual number, i.e., $f/1.4, f/2, f/2.8, \dots, f/22$. (Alternatively, we can say $N = 1.4$, etc.) An easy way to interpret these numbers is to notice that dividing the focal length by the f-number gives us the diameter d , so these are just formulas for the aperture diameter.⁴

Notice that the usual progression for f-numbers is in *full stops*, which are multiples of $\sqrt{2}$, since this corresponds to doubling the area of the entrance pupil each time a smaller f-number is selected. (This doubling is also called changing the exposure by one *exposure value* or EV. It has the same effect on the amount of light reaching the sensor at doubling the exposure duration, e.g., from $1/125$ to $1/250$ —see Exercise 2.5.)

Now that you know how to convert between f-numbers and aperture diameters, you can construct your own plots for the depth of field as a function of focal length f , circle of confusion c , and focus distance z_o , as explained in Exercise 2.4. and see how well these match what you observe on actual lenses such as those shown in Figure 2.21.

Chromatic aberration

Because the index of refraction for glass varies slightly as a function of wavelength, simple lenses suffer from *chromatic aberration*, which is the tendency for light of different colors to focus at slightly different distances (and hence also with slightly different magnification factors), as shown in Figure 2.22. The wavelength-dependent magnification factor can be modeled as a per-color radial distortion (§2.1.5), and hence calibrated using the techniques described in §5.3.5. The wavelength-dependent blur can be calibrated using techniques described in §9.1.3.

In order to reduce chromatic and other kinds of aberrations, most photographic lenses today are *compound lenses* made of different glass elements (with different coatings). Such lenses can no longer be modeled as having a single *nodal point* P through which all of the rays must pass

⁴ This also explains why with zoom lenses, the f-number varies with the current zoom (focal length) setting.

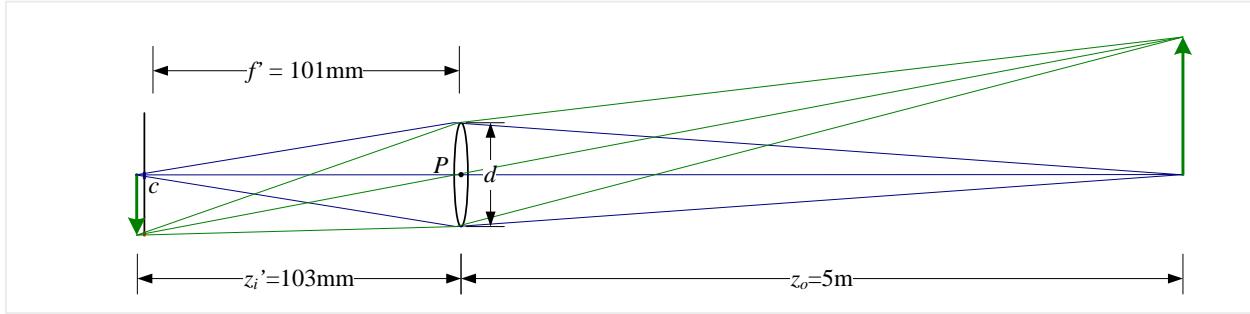


Figure 2.22: *In a lens subject to chromatic aberration, light at different wavelengths (e.g., the green arrow) may get focused with a different focal length f' and hence a different depth z'_i , resulting in both a geometric (in-plane) displacement and a loss of focus.*

(when approximating the lens with a pinhole model). Instead, these lenses have both a *front nodal point*, through which the rays enter the lens, and a *rear nodal point*, through which they leave on their way to the sensor. In practice, only the location of the front nodal point is of interest when performing careful camera calibration, e.g., when determining the point around which to rotate to capture a parallax-free panorama (§8.1.3).

Not all lenses, however, can be modeled as having a single nodal point. In particular very wide angle lenses such as fisheye lenses §2.1.5 and certain *catadioptric* imaging systems consisting of lenses and curved mirrors (Baker and Nayar 1999) do not have a single point through which all of the acquired light rays pass. In such cases, it is preferable to explicitly construct a mapping function (look-up table) between pixel coordinates and 3D rays in space (Gremban *et al.* 1988, Champleboux *et al.* 1992a, Grossberg and Nayar 2001, Sturm and Ramalingam 2004, Tardif *et al.* 2006), as mentioned in §2.1.5.

Vignetting

Another property of real-world lenses is *vignetting*, which is the tendency for the brightness of the image to fall off towards the edge of the image.

Two different kinds of phenomena usually contribute to this effect (Ray 2002). The first is called *natural vignetting* and is due to the foreshortening in the object surface, projected pixel, and lens aperture, as shown in Figure Figure 2.23. Consider the light leaving the object surface patch of size δo located at an *off-axis angle* α . Because this patch is foreshortened with respect to the camera lens, the amount of light reaching the lens is reduced by a factor $\cos \alpha$. The amount of light reaching the lens is also subject to the usual $1/r^2$ fall-off; in this case, the distance $r_o = z_o / \cos \alpha$. The actual area of the aperture through which the light passes is foreshortened by an additional factor $\cos \alpha$, i.e., the aperture as seen from point O is an ellipse of dimensions $d \times d \cos \alpha$. Putting

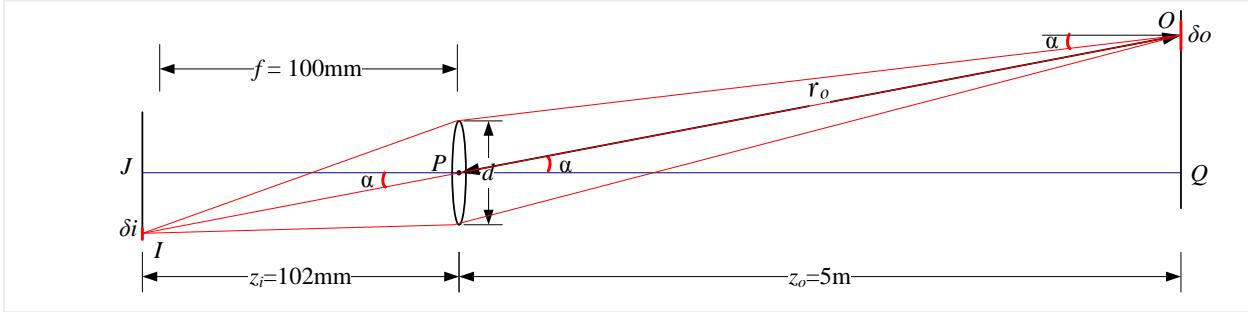


Figure 2.23: *The amount of light hitting a pixel of surface area δi depends on the square of the ratio of the aperture diameter d to the focal length f , as well as the fourth power of the off-axis angle α cosine, $\cos^4 \alpha$.*

all of these factors together, we see that the amount of light leaving O and passing through the aperture on its way to the image pixel located at I is proportional to

$$\frac{\delta o \cos \alpha}{r_o^2} \pi \left(\frac{d}{2} \right)^2 \cos \alpha = \delta o \frac{\pi}{4} \frac{d^2}{z_o^2} \cos^4 \alpha. \quad (2.94)$$

Since triangles ΔOPQ and ΔIPJ are similar, the projected areas of the object surface δo and image pixel δi are in the same (squared) ratio as $z_o : z_i$,

$$\frac{\delta o}{\delta i} = \frac{z_o^2}{z_i^2}. \quad (2.95)$$

Putting these together, we obtain the final relationship between the amount of light reaching pixel i and the aperture diameter d , the focusing distance $z_i \approx f$, and the off-axis angle α ,

$$\delta o \frac{\pi}{4} \frac{d^2}{z_o^2} \cos^4 \alpha = \delta i \frac{\pi}{4} \frac{d^2}{z_i^2} \cos^4 \alpha \approx \delta i \frac{\pi}{4} \left(\frac{d}{f} \right)^2 \cos^4 \alpha, \quad (2.96)$$

which is called the *fundamental radiometric relation* between the scene radiance L and the light (irradiance) E reaching the pixel sensor,

$$E = L \frac{\pi}{4} \left(\frac{d}{f} \right)^2 \cos^4 \alpha \quad (2.97)$$

(Horn 1986, Nalwa 1993, Hecht 2001, Ray 2002). Notice in this equation how the amount of light depends on the pixel surface area (which is why the smaller sensors in point-and-shoot cameras are so much noisier than digital SLRs), the inverse squared of the f-stop $N = f/d$ (2.93), and the fourth power of the $\cos^4 \alpha$ off-axis fall-off, which is the natural vignetting term.

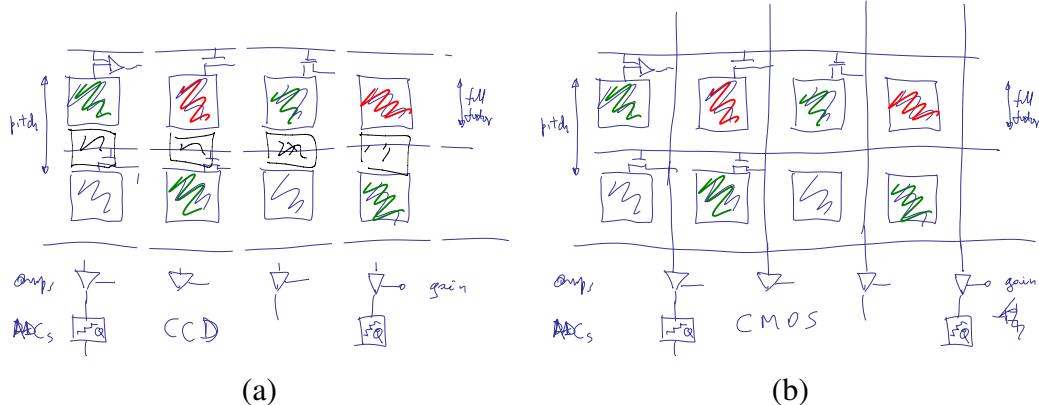


Figure 2.24: Simplified diagrams of image sensors: (a) CCD, where sensed charges are shifted down to a set of amplifiers and analog-to-digital converters; (b) CMOS, where sensed charges are multiplexed onto analog lines. Different cells may have different color filters applied to them.
 [Note: Find some literature on these two to see if this is right. In particular, CMOS may have individual amplifiers at each cell.]

The other major kind of vignetting, called *mechanical vignetting*, is caused by the internal occlusion of rays near the periphery of lens elements in a compound lens, and cannot easily be described mathematically without performing a full ray-tracing of the actual lens design. (See (Kang and Weiss 2000, Zheng *et al.* 2006) for some empirical models that work well in practice.) However, unlike natural vignetting, mechanical vignetting can be decreased by reducing the camera aperture (increasing the f-number). It can also be calibrated (along with natural vignetting) using special devices such as integrating spheres, uniformly illuminated targets, or camera rotation, as discussed in §9.1.2.

2.3 The digital camera

After starting from one or more light sources, reflecting off of one or more surfaces in the world, and passing through the camera's optics (lenses), light finally reaches the imaging sensor. How are the photons arriving at this sensor converted into the digital (R,G,B) values that we observe when we look at a digital image? In this section, we develop a simple model that accounts for the most important effects such as exposure (gain and shutter speed), non-linear mappings, sampling and aliasing, and noise.

Light falling on an imaging sensor is usually picked up by an *active sensing area*, integrated for the duration of the exposure (usually expressed as the shutter speed, e.g., $\frac{1}{125}$, $\frac{1}{60}$, $\frac{1}{30}$ of a second), and then passed to a set of *sense amplifiers* (Figure 2.24).

The two main kinds of sensors used in digital still and video cameras today are CCD (charge coupled device) and CMOS (complementary metal oxide on silicon). In a CCD, photons are accumulated in each active *well* during the exposure time. In a subsequent *transfer* phase, the charges are transferred from well to well in a kind of “bucket brigade” until they are deposited at the sense amplifiers, which then amplify the signal and pass it to an analog-to-digital converter (ADC).⁵ Older CCD sensor were prone to *blooming* when charges from one over-exposed pixel spilled into adjacent ones, but most newer CCDS have anti-blooming technology (“troughs” where the excess charge can spill over).

In CMOS, the photos hitting the sensor directly affect the conductivity (or gain) of a photosensitive transistor. The resulting voltage is then sensed using a multiplexing scheme, and amplified and sampled as with a CCD. [*Note: This is probably wrong. Read up about this and fix it up.*] Traditionally, CCD sensors outperformed CMOS in quality sensitive applications such as digital SLRs, while CMOS was better for low-power applications, but today, CMOS is used in most digital cameras.

The main factors affecting the performance of a digital image sensor are the shutter speed, sampling pitch, fill factor, chip size, analog gain, sensor noise, and the resolution (and quality) of the analog-to-digital converter. Many of the actual values for these parameters can be read off from the EXIF tags embedded with digital images [*Note: need a cite for EXIF here*], while others can be obtained from the camera manufacturers’ spec. sheets or from camera review or calibration Web sites such as <http://dpreview.com/> or [*Note: PTLens???*]

Shutter speed. The shutter speed (exposure time) directly controls the amount of light reaching the sensor, and hence determines if images are under or over-exposed. (For bright scenes where a large aperture and/or slow shutter speed are desired to get a shallow depth of field and/or motion blur, *neutral density filters* are sometimes used by photographers.) For dynamic scenes, the shutter speed also determines the amount of *motion blur* in the resulting picture. Usually, a higher shutter speed (less motion blurs) makes subsequent analysis easier (see §9.3 for techniques to remove such blur). However, when video is being captured for display, some motion blur may be desirable to avoid stroboscopic effects.

Sampling pitch. The sampling pitch is the physical spacing between adjacent sensor cells on the imaging chip (Figure 2.24). A sensor with a smaller sampling pitch has a higher *sampling density*, and hence provides a higher *resolution* (in terms of pixels) for a given active chip area. However,

⁵ In digital still cameras, a complete frame is captured and then read out sequentially at once. In older video cameras, the even fields (lines) are scanned first, followed by the odd fields, in a process that is called *interlacing* §13.1.

a smaller pitch also means that each sensor has a smaller area, and hence it cannot accumulate as many photons and so is not as *light sensitive* (and hence more prone to noise).

Fill factor. The fill factor is the active sensing area size as a fraction of the theoretically available sensing area (the product of the horizontal and vertical sampling pitches). Higher fill factors are usually preferable, as they result in more light capture and less *aliasing* (see §2.3.1). However, these must be balanced with the need to place additional electronics in between the active sense areas. Figure 2.24 shows a hypothetical sensor where the active area only takes up one-quarter of the available space, and hence has a fill factor of 0.25. More typical numbers for camera are... [Note: DPReview does not seem to have these. Do the camera specs, or PTLens have them?]. The fill factor can also be determined empirically using a photometric camera calibration process §9.1.3.

Chip size. Video and point-and-shoot cameras have traditionally used small chip areas ($\frac{1}{4}$ inch to $\frac{1}{2}$ inch sensors⁶), while digital SLR (single lens reflex) cameras try to come closer to the traditional size of a 35mm film frame.⁷ When overall device size is not important, having a larger chip size is preferable, since each sensor cell can be more photo-sensitive. (For compact cameras, a smaller chip means that all of the optics can be shrunk down proportionately.) However, larger chips are more expensive to produce, not only because fewer chips can be packed into each wafer, but also because the probability of a chip defect goes up linearly with the chip area.

Analog gain. Before analog-to-digital conversion, the sensed signal is usually boosted by a *sense amplifier*. In video cameras, the gain on these amplifiers was traditionally controlled by *automatic gain control* (AGC) logic, which would adjust these values to obtain a good overall exposure. In newer digital still cameras, the user now has some additional control over this gain through the *ISO setting*, which is typically expressed in ISO standard units like 100, 200, or 400. Since the automated exposure control in most cameras also adjusts the aperture and shutter speed, setting the ISO manually removes one degree of freedom from the camera's control, just as manually specifying aperture and shutter speed does. In theory, a higher gain allows the camera to perform better under low light conditions (less motion blur due to long exposure times when the aperture is already maxed out). In practice, however, higher ISO settings usually amplify the *sensor noise*.

⁶ These numbers refer to the “tube diameter” of the old vidicon tubes used in video cameras <http://www.dpreview.com/learn/?key=sensor%20sizes>. The 1/2.5” sensor on the Canon SD800 camera actually measures 5.76mm × 4.29mm, i.e., $\frac{1}{6}$ th the size (on side) of a full frame DSLR sensor.

⁷ When a DSLR chip does not fill the full frame, it results in a *multiplier effect* on the lens focal length. For example, a chip that is only 0.6 the dimension of a 35mm frame will make a 50mm lens image the same angular extent as a $50/0.6 = 50 \times 1.6 = 80$ mm lens, as demonstrated in (2.59).

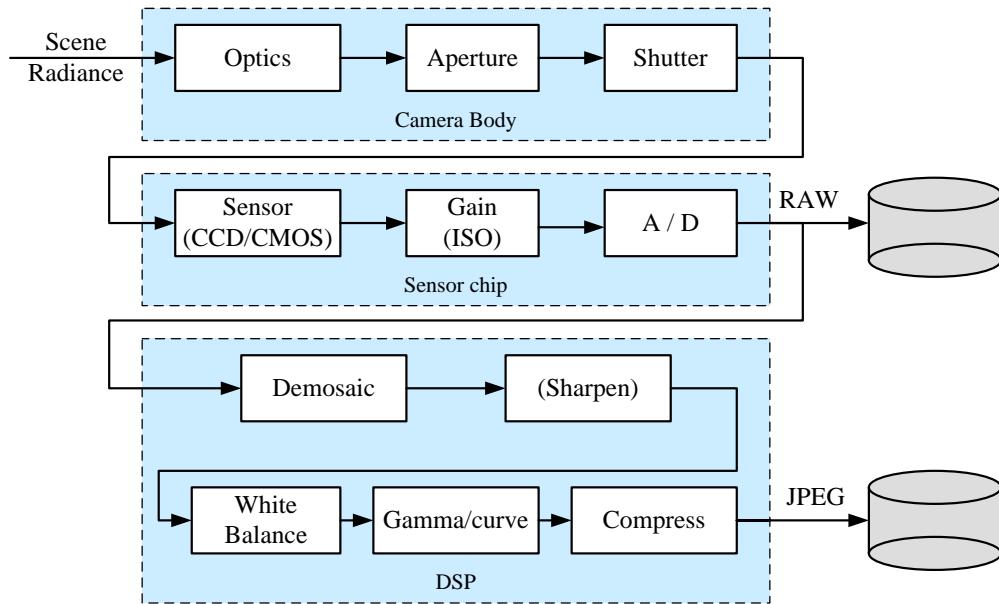


Figure 2.25: *Image sensing pipeline, showing the various sources of noise as well as the typical digital post-processing steps.*

Sensor noise. Throughout the whole sensing process, noise is being added due to various sources, which may include *fixed pattern noise*, *dark current noise*, *shot noise*, *amplifier noise* and *quantization noise* (Healey and Kondepudy 1994, Tsin *et al.* 2001). The final amount of noise present in a sampled image depends on all of these quantities, as well as the incoming light (controlled by the scene radiance and aperture), the exposure time, and the sensor gain. Liu *et al.* (2008) use this model, along with an empirical database of camera response functions (CRFs) obtained from (Grossberg and Nayar 2004), to estimate the *noise level function* (NLF) for a given image, which predicts the overall noise variance at a given pixel as a function of its brightness (a separate NLF is estimated for each color channel). An alternative approach, when you have access to the camera before taking pictures, is to pre-calibrate the NLF by taking repeated shots of a scene containing a variety of colors and luminances such as a Macbeth Color Chart [Note: add cite here]. (When estimating the variance, be sure to throw away or downweight pixels with large gradients, as small shifts between exposures will affect the sensed values at such pixels.) Unfortunately, the pre-calibration process may have to be repeated for different exposure times and gain settings because of the complex interactions occurring within the sensing system.

In practice, most computer vision algorithms, such as image denoising, edge detection, and stereo matching, all benefit from at least a rudimentary estimate of the noise level. Barring the ability to pre-calibrate the camera or to take repeated shots of the same scene, the simplest approach is to look for regions of near-constant value and to estimate the noise variance in such regions (Liu

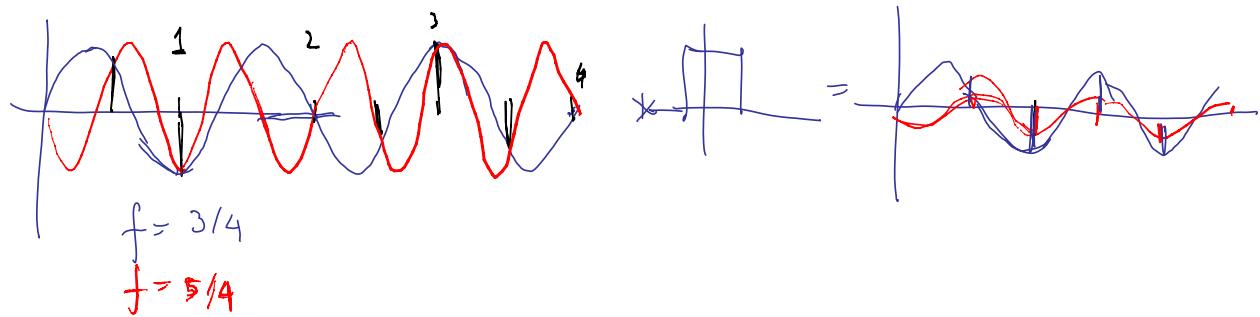


Figure 2.26: *Aliasing of a one-dimensional signal. The blue sine wave at $f = \frac{3}{4}$ and the red sine wave at $f = \frac{5}{4}$ have the same digital samples, when sampled at $f = 2$. Even after convolution with a 100% fill factor box filter, the two signals, while no longer of the same magnitude, are still aliased.*

et al. 2008).

ADC resolution. The final step in the analog processing chain occurring within an imaging sensor is the *analog to digital conversion* (ADC). While a variety of techniques can be used to implement this process, the two quantities of interest are the *resolution* of this process (how many bits it yields) and its noise level (how many of these bits are useful in practice). For most cameras, the number of bits quoted (8 bits for compressed JPEG images, and a nominal 16 bits for the RAW formats provided by some DSLRs) exceeds the actual number of usable bits. The best way to tell is to simply calibrate the noise of a given sensor, e.g., by taking repeated shots of the same scene and plotting the estimated noise as a function of brightness (Exercise 2.6).

Digital post-processing. Once the irradiance values arriving at the sensor have been converted to digital bits, most cameras perform a variety of *digital signal processing* (DSP) operations to enhance the image before compressing and storing the pixel values. These include color filter (CFA) array de-mosaicing and whitepoint setting, and the mapping of the luminance values through a *gamma function* to increase the perceived dynamic range of the signal. I will cover these topics shortly in §2.3.2, but before I do, I want to return to the topic of aliasing, which I mentioned above in connection with sensor array fill factors.

2.3.1 Sampling and aliasing

What happens when a field of light impinging on the image sensor falls onto the active sense areas in the imaging chip? The photons arriving at each active cell are integrated and then digitized, as

shown in Figure 2.24. However, if the fill factor on the chip is small, and the signal is not otherwise *band-limited*, visually unpleasing *aliasing* can occur.

To explore the phenomenon of aliasing, let us first look at a one-dimensional signal (Figure 2.26), in which we have two sine waves, one at a frequency of $f = 3/4$ and the other at $f = 5/4$. If we sample these two signals at a frequency of $f = 2$, we see that they produce the same samples (shown in black), and so we say that they are *aliased*.⁸ Why is this a bad effect? In essence, we can no longer reconstruct the original signal, since we do not know which of the two original frequencies was present.

In fact, Shannon's Sampling Theorem shows that the minimum sampling (Oppenheim and Schafer 1996, Oppenheim *et al.* 1999) rate required to reconstruct a signal from its instantaneous samples must be at least twice the highest frequency,⁹

$$f_s \geq 2f_{\max}. \quad (2.98)$$

The maximum frequency in a signal is known as the *Nyquist frequency*, and the inverse of the minimum sampling frequency $r_s = 1/f_s$ is known as the *Nyquist rate*.

However, you may ask, since an imaging chip actually *averages* the light field over a finite area, are the results on point sampling still applicable? Averaging over the sensor area does tend to attenuate some of the higher frequencies. However, even if the fill factor is 100%, as in the right hand side of (Figure 2.26), frequencies above the Nyquist limit (half the sampling frequency) still produce an aliased signal, although with a smaller magnitude than the corresponding band-limited signals.

A more convincing argument as to why aliasing is bad can be seen by downsampling a signal using a poor quality filter such as a box (square) filter. Figure 2.27 shows a high-frequency *chirp* image (so called because the frequencies increase over time), along with the results of sampling it with a 25% fill-factor area sensor, a 100% fill-factor sensor, and a high-quality 9-tap filter. Additional examples of downsampling (*decimation*) filters can be found in §3.4.1 and Figures 3.31–3.33.

The best way to predict the amount of aliasing that an imaging system (or even an image processing algorithm) will produce is to estimate the *point spread function* (PSF), which represents the response of a particular pixel sensor to an ideal point light source. The PSF is a combination (convolution) of the blur induced by the optical system (lens) and the finite integration area of a chip sensor.¹⁰

⁸ An alias is an alternate name for someone, so the sampled signal corresponds to two different *aliases*.

⁹ The actual theorem states that f_s must be at least twice the signal *bandwidth*, but since we are not dealing with modulated signals such as radio waves during image capture, the maximum frequency suffices.

¹⁰ Imaging chips also usually interpose an optical *anti-aliasing filter* just before the imaging chip to reduce or control the amount of aliasing.

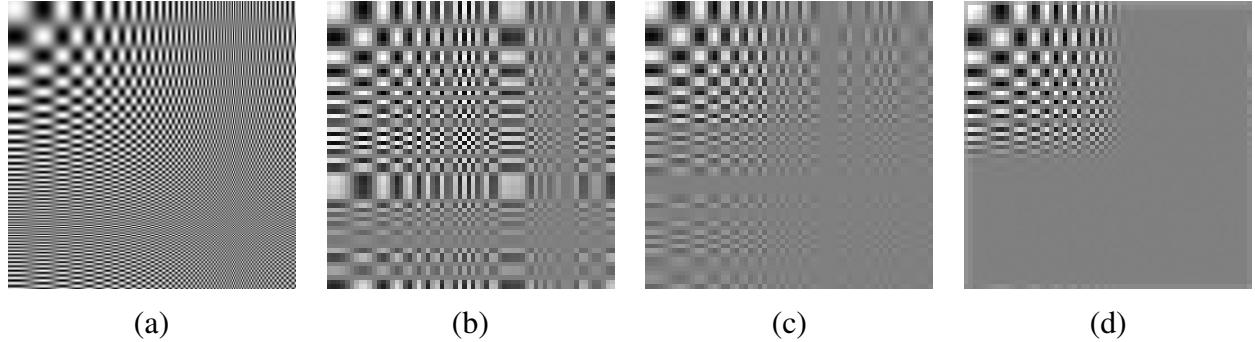


Figure 2.27: *Example of 2D image aliasing: (a) original full-resolution image; (b) downsampled 4× with a 25% fill factor box filter; (c) downsampled 4× with a 100% fill factor box filter; (d) downsampled 4× with a high-quality 9-tap filter. Notice how the higher frequencies are aliased into visible frequencies with the lower quality filters, while the 9-tap filter completely removes these higher frequencies.*

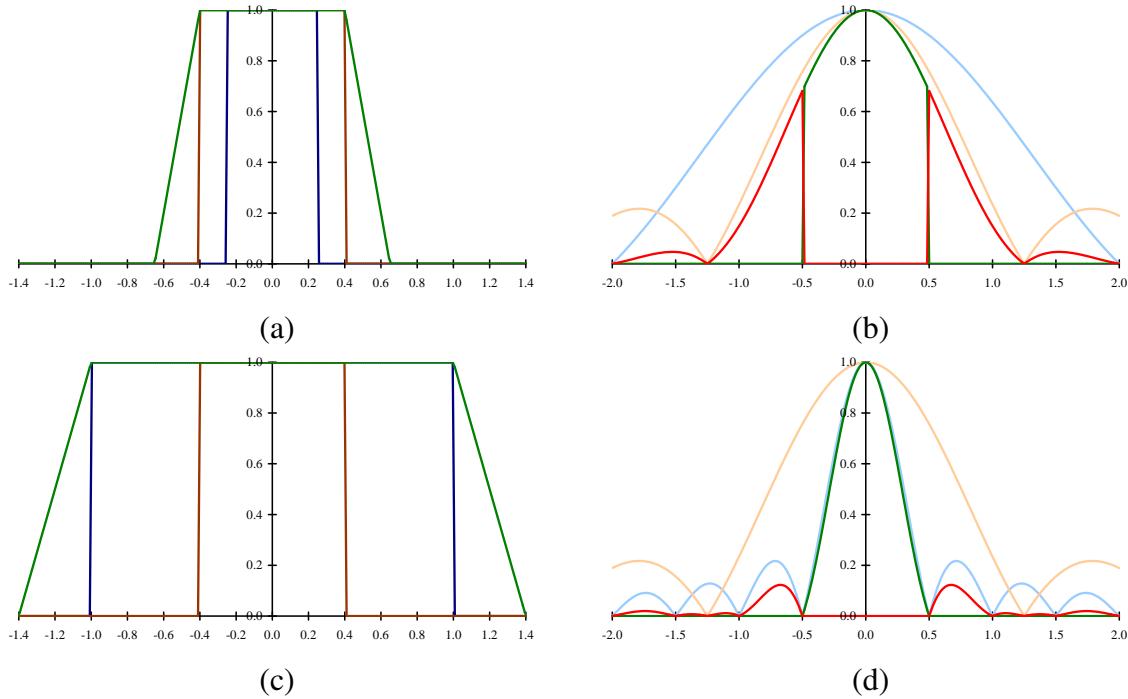


Figure 2.28: *Sample point spread functions (PSF). The diameter of the blur disc (blue) in (a) is equal to half the pixel spacing, while the diameter in (c) twice the pixel spacing. The horizontal fill factor of the sensing chip is 80% and is shown in brown. The convolution of these two kernels gives the point spread function, shown in green. The Fourier response of the PSF (the MTF) is plotted in (b) and (d), The area above the Nyquist frequency where aliasing occurs is shown in red.*

If we know the blur function of the lens and the fill factor (sensor area shape and spacing) for the imaging chip (plus, optionally, the response of the anti-aliasing filter), we can convolve these (as described in §3.2.1) to obtain the PSF. Figure 2.28a shows the one-dimensional cross-section of a PSF for a lens whose blur function is assumed to be a disc of a radius equal to the pixel spacing s plus a sensing chip whose horizontal fill factor is 80%. Taking the Fourier transform of this PSF (§3.3), we obtain the *modulation transfer function* (MTF), from which we can estimate the amount of aliasing as the area of the Fourier magnitude outside the $f \leq f_s$ Nyquist frequency.¹¹ If we de-focus the lens so that the blur function has a radius of $2s$ (Figure 2.28b), we see that the amount of aliasing decreases significantly, but so does the amount of image detail (frequencies closer to $f = f_s$).

Under laboratory conditions, the PSF can be estimated (to pixel precision) by looking at a point light source such as a pin hole in a black piece of cardboard lit from behind. However, this PSF (the actual image of the pinhole) is only accurate to a pixel resolution, and while it can model larger blur (such as caused by defocus), it cannot model the sub-pixel shape of the PSF, and hence predict the amount of aliasing.

An alternative technique, described in §9.1.3, is to look at a calibration pattern (e.g., one consisting of slanted step edges (Reichenbach *et al.* 1991, Williams and Burns 2001, Joshi *et al.* 2008)) whose ideal appearance can be re-synthesized to sub-pixel precision.

In addition to occurring during image acquisition, aliasing can also be introduced in various image processing operations such as resampling, upsampling, and downsampling. Sections 3.3–3.4.1 discuss these issues, and show how careful selection of filters can reduce the amount of aliasing that operations inject.

2.3.2 Color

In §2.2 on photometric image formation, we saw how lighting and surface reflections are functions of wavelength. When the incoming light hits the imaging sensor, light from different parts of the spectrum is somehow integrated into the discrete RGB (red, green, and blue) color values that we see in a digital image. How does this process work, and how can we analyze and manipulate color values?

You probably recall from your childhood days the magical process of mixing paint colors to obtain new ones. You may recall that blue+yellow makes green, red+blue makes purple, and red+green makes brown. If you revisited this topic at a later age, you may have learned that the proper *subtractive* primaries are actually cyan (a light blue-green), magenta (pink), and yellow

¹¹ The complex Fourier transform of the point spread function (PSF) is actually called the *optical transfer function* (OTF) (Williams 1999). Its magnitude is called the *modulation transfer function* (MTF) and its phase is called the *phase transfer function* (PTF).

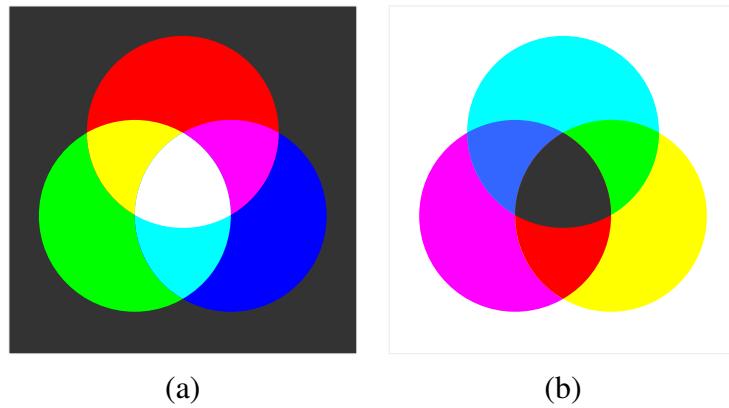


Figure 2.29: Primary and secondary colors: (a) additive colors red, green, and blue can be mixed to produce cyan, magenta, yellow, and white; (b) subtractive colors cyan, magenta, and yellow can be mixed to produce red, green, and blue, as well as black.

(Figure 2.29b), although black is also often used in four-color printing (CMYK). (If you ever subsequently took any painting classes, you learned that colors can have even more fanciful names such as alizarin crimson, cerulean blue, and chartreuse.) The subtractive colors are called subtractive because pigments in the paint absorb certain wavelengths in the color spectrum.

Later on in high-school, you may have learned about the *additive* primary colors red, green, and blue (RGB), and how they can be added (with a slide projector, or on a computer monitor) to produce cyan, magenta, yellow, white, and all the other colors we typically see on our TV sets and monitors (Figure 2.29a).

Through what process is it possible for two different colors, such as red and green, to interact to produce a third color like yellow? Are the wavelengths somehow mixed up to produce a new wavelength?

You probably know that the correct answer has nothing to do with physically mixing wavelengths. Instead, the existence of three primaries is a result of the *tri-stimulus* (or *tri-chromatic*) nature of the human visual system, since we have three different kinds of cones, each of which responds selectively to a different portion of the color spectrum (Glassner 1995, Wyszecki and Stiles 2000, Reinhard *et al.* 2005). (Note that for machine vision application such as remote sensing and terrain classification, it is preferable to use many more wavelengths. Similarly, surveillance applications can often benefit from sensing in the near-infrared (NIR) range.)

CIE RGB and XYZ

To test and quantify the tri-chromatic theory of perception, we can attempt to reproduce all *monochromatic* (single wavelength) colors as a mixture of three suitably chosen primaries. (Pure wave-

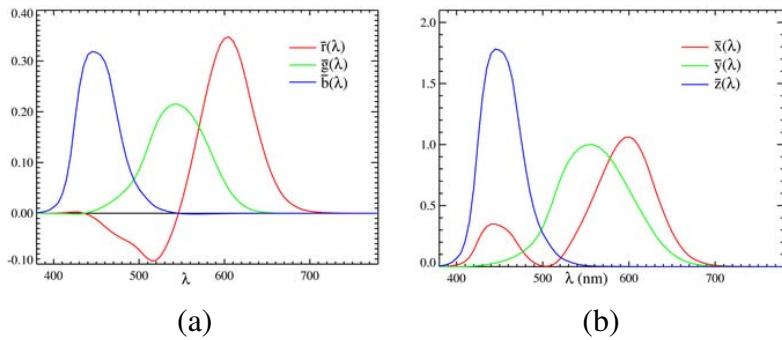


Figure 2.30: Standard CIE color matching functions: (a) $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, $\bar{b}(\lambda)$ color spectra obtained from color matching pure colors to the R=700.0nm, G=546.1nm, and B=435.8nm primaries; (b) $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$ color matching functions, which are linear combinations of the $(\bar{r}(\lambda), \bar{g}(\lambda), \bar{b}(\lambda))$ spectra.

[Note: Replace the above figures (from Wikipedia) with my own plots produced with data from (Glassner 1995, §G.2) or <http://www-cvrl.ucsd.edu/>]

length light can be obtained using either a prism or specially manufactured color filters.) In the 1930s, the CIE (Commission Internationale d'Eclairage) standardized the RGB representation by performing such *color matching* experiments using the primary colors of red (700.0nm wavelength), green (546.1nm), and blue (435.8nm).

Figure 2.30 shows the results of performing these experiments with a *standard observer*, i.e., averaging perceptual results over a large number of subjects. You will notice that for certain pure spectra in the blue-green range, a *negative* amount of red light has to be added, i.e., a certain amount of red has to be added to the color being matched in order to get a color match. These results also provided a simple explanation for the existence of *metamers*, which are colors with different spectra that are perceptually indistinguishable. (Note that two fabrics or paint colors that are metamers under one light may no longer be so under different lighting.)

Because of the problem associated with mixing negative light, the CIE also developed a new color space called XYZ, which contains all of the pure spectral colors within its positive octant. (It also maps the Y axis to the *luminance*, i.e., perceived relative brightness, and maps pure white to a diagonal (equal-valued) vector.) The transformation from RGB to XYZ is given by

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \quad (2.99)$$

[Note: Check these numbers with <http://www-cvrl.ucsd.edu/> and/or (Wyszecki and Stiles 2000).] While the official definition of the CIE XYZ standard has the matrix normalized so that the Y value corresponding to pure red is 1, a more commonly used form is to omit the leading

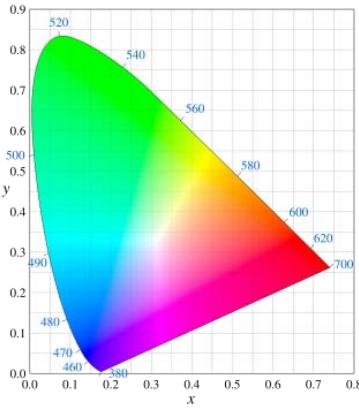


Figure 2.31: *CIE chromaticity diagram, showing colors and their corresponding (x, y) values. Pure spectral colors are arranged around the outside of the curve.*

[Note: Replace the above figures (from Wikipedia) with my own plot produced with data from ([Glassner 1995](#)) or <http://www-cvrl.ucsd.edu/>]

fraction, so that the second row sums up to one, i.e., the RGB triplet $(1, 1, 1)$ maps to a Y value of 1. Linearly blending the $(\bar{r}(\lambda), \bar{g}(\lambda), \bar{b}(\lambda))$ curves in Figure 2.30a according to (2.99), we obtain the resulting $(\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda))$ curves shown in Figure 2.30b. Notice how all three spectra (color matching functions) now have only positive values, and how the $\bar{y}(\lambda)$ curves matches that of the luminance perceived by humans.

If we divide the XYZ values by the sum of X+Y+Z, we obtain the *chromaticity coordinates*

$$x = \frac{X}{X + Y + Z}, \quad y = \frac{Y}{X + Y + Z}, \quad z = \frac{Z}{X + Y + Z}, \quad (2.100)$$

which sum up to 1. The chromaticity coordinates discard the absolute intensity of a given color sample and just represents its pure color. If we sweep the monochromatic color λ parameter in Figure 2.30b from $\lambda = 380\text{nm}$ to $\lambda = 800\text{nm}$, we obtain the familiar *chromaticity diagram* shown in Figure 2.31. This figure shows the (x, y) value for every color value perceivable by most humans. (Of course, the CMYK reproduction process in this book does not actually span the whole *gamut* of perceivable colors.) The outer curved rim represents where all of the pure monochromatic color values map to in (x, y) space, while the lower straight line, which connects the two endpoints, is known as the *purple line*.

A convenient representation for color values, when we want to tease apart luminance and chromaticity is therefore Yxy (luminance plus the two most distinctive chrominance components).

L*a*b* color space

While the XYZ color space has many convenient properties, including the ability to separate luminance from chrominance, it does not actually predict how well humans perceive *differences* in color or luminance.

Because the response of the human visual system is roughly logarithmic (we can perceive *relative* luminance differences of about 1%), the CIE defined a non-linear re-mapping of the XYZ space called L*a*b* (also sometimes called CIELAB), where differences in luminance or chrominance are more perceptually uniform.

The L* component of *lightness* is defined as

$$L^* = 116f\left(\frac{Y}{Y_n}\right), \quad (2.101)$$

where Y_n is the luminance value for nominal white (Fairchild 1998) and

$$f(t) = \begin{cases} t^{1/3} & t > \delta^3 \\ t/(3\delta^2) + 2\delta/3 & \text{else,} \end{cases} \quad (2.102)$$

is a finite-slope approximation to the cube root with $\delta = 6/29$. The resulting 0 . . . 100 scale roughly measures equal amounts of lightness perceptibility.

In a similar fashion, the a* and b* components are defined as

$$a^* = 500 \left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right] \text{ and } b^* = 200 \left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right], \quad (2.103)$$

where again, (X_n, Y_n, Z_n) is the measured whitepoint. Figure 2.34i–k show the L*a*b* representation for a sample color image.

Color cameras

While the preceding discussion tells us how we can uniquely describe the perceived tri-stimulus description of any color (spectral distribution), it does not tell us how RGB still and video cameras actually work. Do they just measure the amount of light at the nominal wavelengths of red (700.0nm), green (546.1nm), and blue (435.8nm)? Do color monitors just emit exactly these wavelengths, and if so, how can they emit negative red light to reproduce colors in the cyan range?

In fact, the design of RGB video cameras has historically been based around the availability of colored phosphors that go into television sets. When standard-definition color television was invented (NTSC), a mapping was defined between the RGB values that would drive the three color guns in the CRT and the XYZ values that unambiguously define perceived color (this standard was

called ITU-R BT.601). With the advent of HDTV and newer monitor, a new standard called ITU-R BT.709 was created, which specifies the XYZ values of each of the color primaries,

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R_{709} \\ G_{709} \\ B_{709} \end{bmatrix}. \quad (2.104)$$

In practice, each color camera integrates light according to the *spectral response function* of its red, green, and blue sensors,

$$\begin{aligned} R &= \int L(\lambda)S_R(\lambda)d\lambda, \\ G &= \int L(\lambda)S_G(\lambda)d\lambda, \\ B &= \int L(\lambda)S_B(\lambda)d\lambda, \end{aligned} \quad (2.105)$$

where $L(\lambda)$ is the incoming spectrum of light at a given pixel and $\{S_R(\lambda), S_G(\lambda), S_B(\lambda)\}$ are the red, green, and blue *spectral sensitivities* of the corresponding sensors.

Can we tell what spectral sensitivities the cameras actually have? Unless the camera manufacturer provides us with this data, or we observe the response of the camera to a whole spectrum of monochromatic lights, these sensitivities are *not* specified by a standard such as BT.709. Instead, all that matters is that the tri-stimulus values for a given color produce the specified RGB values. The manufacturer is free to use sensors with sensitivities that do not match the standard XYZ definitions, so long as they can later be converted (through a linear transform) to the standard colors.

Similarly, while TV and computer monitors are supposed to produce RGB values as specified by (2.104), there is no reason that they cannot use digital logic to transform the incoming RGB values into different signals to drive each of the color channels. Properly calibrated monitors make this information available to software applications that perform *color management*, so that colors in real life, on the screen, and on the printer all match as closely as possible.

Color filter arrays

While early color TV cameras used 3 different *vidicons* (tubes) to perform their sensing, and later cameras used 3 separate RGB sensing chips, most of today's digital still and video cameras cameras use a *color filter array* (CFA), where alternating sensors are covered by different colored filters (Figure 2.24). (A newer chip design by Foveon <http://foveon.com> [Note: check] stacks the red, green, and blue sensors beneath each other, but it has not yet gained widespread adoption.)

The most commonly used pattern in color cameras today is the *Bayer pattern*, which places green filters over half of the sensors (in a checkerboard pattern), and red and blue filters over the

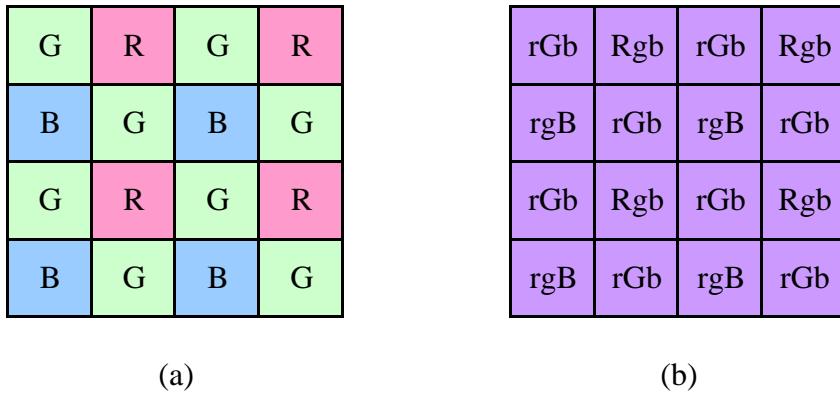


Figure 2.32: *Bayer RGB pattern*: (a) color filter array layout; (b) interpolated pixel values, with unknown (guessed) values shown as lower case.

remaining ones (Figure 2.32). The reason that there are twice as many green filters as red and blue is because the luminance signal is mostly determined by green values, and the visual system is much more sensitive to high frequency detail in luminance than in chrominance (a fact that is exploited in color image compression §2.3.3). The process of *interpolating* the missing color values so that we have valid RGB values at all the pixels is known as *de-mosaicing* and will be covered in detail in §9.3.1.

Similarly, color LCD monitors typically use alternating stripes of red, green, and blue filters placed in front of each liquid crystal active area to simulate the experience of a full color display. As before, because the visual system has higher resolution (acuity) in luminance than chrominance, it is possible to digitally pre-filter RGB (and monochrome) images to enhance the perception of crispness (Betrisey *et al.* 2000, Platt 2000).

Color balance

Before encoding the sensed RGB values, most cameras perform some kind of *color balancing* operation in an attempt to move the whitepoint of a given image closer to pure white (equal RGB values). If the color system and the illumination are the same (the BT.709 system uses the daylight illuminant D₆₅ as its reference white), the change may be minimal. However, if the illuminant is strongly colored, such as incandescent indoor lighting (which generally results in a yellow or orange hue), the compensation can be quite significant.

A simple way to perform color correction is to multiply each of the RGB values by a different factor (i.e., to apply a diagonal matrix transform to the RGB color space). More complicated transforms, which are sometimes the result of mapping to XYZ space and back, actually perform

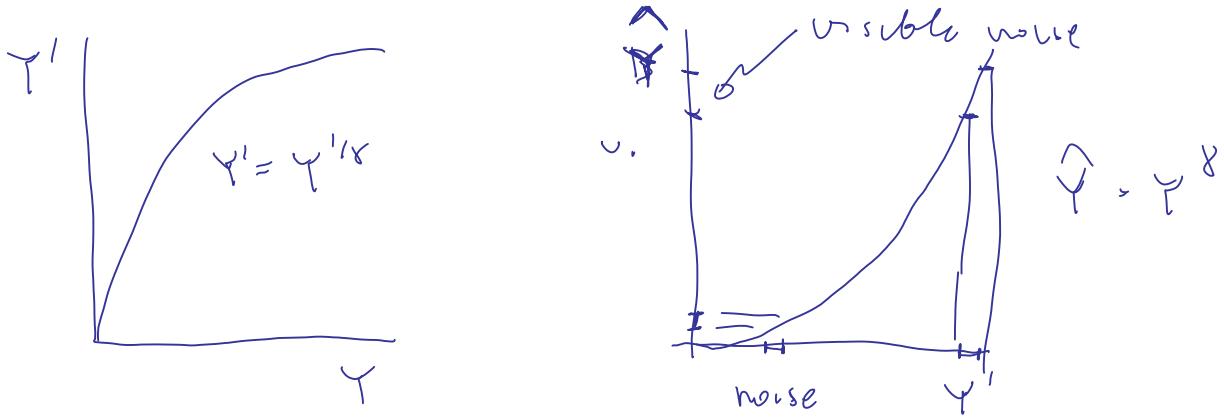


Figure 2.33: *Gamma compression:* (a) The relationship between the input signal luminance Y and the transmitted signal Y' is given by $Y' = Y^{1/\gamma}$. (b) At the receiver, the signal Y' is exponentiated by the factor γ , $\hat{Y} = Y'^{\gamma}$. Noise introduced during transmission is squashed in the dark regions, which corresponds to the more noise-sensitive region of the visual system.

[Note: Generate a proper version of this figure.]

a *color twist*, i.e., use a general 3×3 color transform matrix.¹² Exercise 2.8 has you explore some of these issues.

Gamma

In the early days of black and white television, the phosphors in the CRT (cathode ray tube) used to display the TV signal responded non-linearly to their input voltage. The relationship between the voltage and the resulting brightness was characterized by a number called *gamma* (γ), since the formula was roughly

$$B = V^\gamma, \quad (2.106)$$

with a γ of about 2.2. To compensate for this effect, the electronics in the TV camera would pre-map the sensed luminance Y through an inverse gamma,

$$Y' = Y^{\frac{1}{\gamma}}, \quad (2.107)$$

with a typical value of $\frac{1}{\gamma} = 0.45$.

The mapping of the signal through this non-linearity before transmission had a beneficial side effect: noise added during transmission (remember, these were analog days!) would get reduced

¹² Those of you old enough to remember the early days of color television will naturally think of the *hue* adjustment on the television set, which could produce truly bizarre results.

(after applying the gamma at the receiver) in the darker regions of the signal where it was more visible (Figure 2.33).¹³ (Remember that our visual system is roughly sensitive to relative differences in luminance.)

When color television was invented, it was decided to separately pass the red, green, and blue signals through the same gamma non-linearity before combining them for encoding (see the description of YIQ and YUV below). Today, even though we no longer have analog noise in our transmission systems, signals are still quantized during compression (§2.3.3), so applying inverse gamma to sensed values is still useful.

Unfortunately, for both computer vision and computer graphics, the presence of gamma in images is often problematic. For example, the proper simulation of radiometric phenomena such as shading §2.2 (2.83) occurs in a linear radiance space. Once all of the computations have been performed, the appropriate gamma should be applied before display. Unfortunately, many computer graphics systems (such as shading models) operate directly on RGB values and display these values directly. (Fortunately, newer color imaging standards such as the 16-bit scRGB use a linear space, which makes this less of a problem (Glassner 1995).)

In computer vision, the situation can be even more daunting. The accurate determination of surface normals using a technique such as photometric stereo §11.1.1 requires that the measurements be in a linear space of intensities. Therefore, it is imperative when performing detailed quantitative computations such as these to first undo the gamma in the sensed color values. For other vision applications, however, such as feature detection or the matching of signals in stereo and motion estimation, this linearization step is often not necessary. In fact, determining whether undoing gamma is necessary can take some careful thinking, e.g., in the case of compensating for exposure variations in image stitching (Exercise 2.7).

If all of these processing steps sound confusing to model, they are. Exercise 2.10 has you try to tease apart some of these phenomena using empirical investigation, i.e., taking pictures of color charts and comparing the RAW and JPEG compressed color values.

Other color spaces

While RGB and XYZ are the primary color spaces used to describe the spectral content (and hence tri-stimulus response) of color signals, a variety of other representations have been developed both in video and still image coding and in computer graphics.

The earliest color representation developed for video transmission was the YIQ standard developed for NTSC video in North America, and the closely related YUV standard developed for PAL in Europe. In both of these cases, it was desired to have a *luma* channel Y (so called since it

¹³ A related technique called *companding* was the basis of the Dolby noise reduction systems used with audio tapes.

only roughly mimics true luminance) that would be comparable to the regular black-and-white TV signal, along with two lower frequency *chroma* channels.

In both systems, the Y signal (or more appropriately, the Y' luma signal since it is gamma compressed) is obtained from

$$Y'_{601} = 0.299R' + 0.587G' + 0.114B', \quad (2.108)$$

where R'G'B' is the triplet of gamma compressed color components. When using the newer color definitions for HDTV in BT.709, the formula is

$$Y'_{709} = 0.2125R' + 0.7154G' + 0.0721B'. \quad (2.109)$$

The UV components are derived from scaled versions of $(B' - Y')$ and $(R' - Y')$, namely,

$$U = 0.492111(B' - Y') \text{ and } V = 0.877283(R' - Y'), \quad (2.110)$$

whereas the IQ components are the UV components rotated through an angle of 33° . In composite (NTSC and PAL) video, the chroma signals were then low-pass filtered horizontally before being modulated and superimposed on top of the Y' luma signal. Backward compatibility was achieved by having older black-and-white TV sets effectively ignore the high-frequency chroma signal (because of slow electronics), or, at worst, superimposing it as a high-frequency pattern on top of the main signal.

While these conversions were important in the early days of computer vision, when frame grabbers would directly digitize the composite TV signal, today, all digital video and still image compression standards are based on the newer YCbCr conversion. YCbCr is closely related to YUV (the C_b and C_r signals carry the blue and red color difference signals, and have more useful mnemonics than UV), but uses different scale factors to fit within the 8-bit gamut available with digital signals.

For video, the Y' signal is re-scaled to fit within the [16 . . . 235] range of values, while the Cb and Cr signals are scaled to fit within [16 . . . 240] (Gomes and Velho 1997, Fairchild 1998). For still images, the JPEG standard uses the full 8-bit range with no reserved values,

$$\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}, \quad (2.111)$$

where the R'G'B' values are the 8-bit gamma compressed color components (i.e., the actual RGB values we obtain when we open up or display a JPEG image). For most applications, this formula is not that important, since your image reading software will directly provide you with the 8-bit

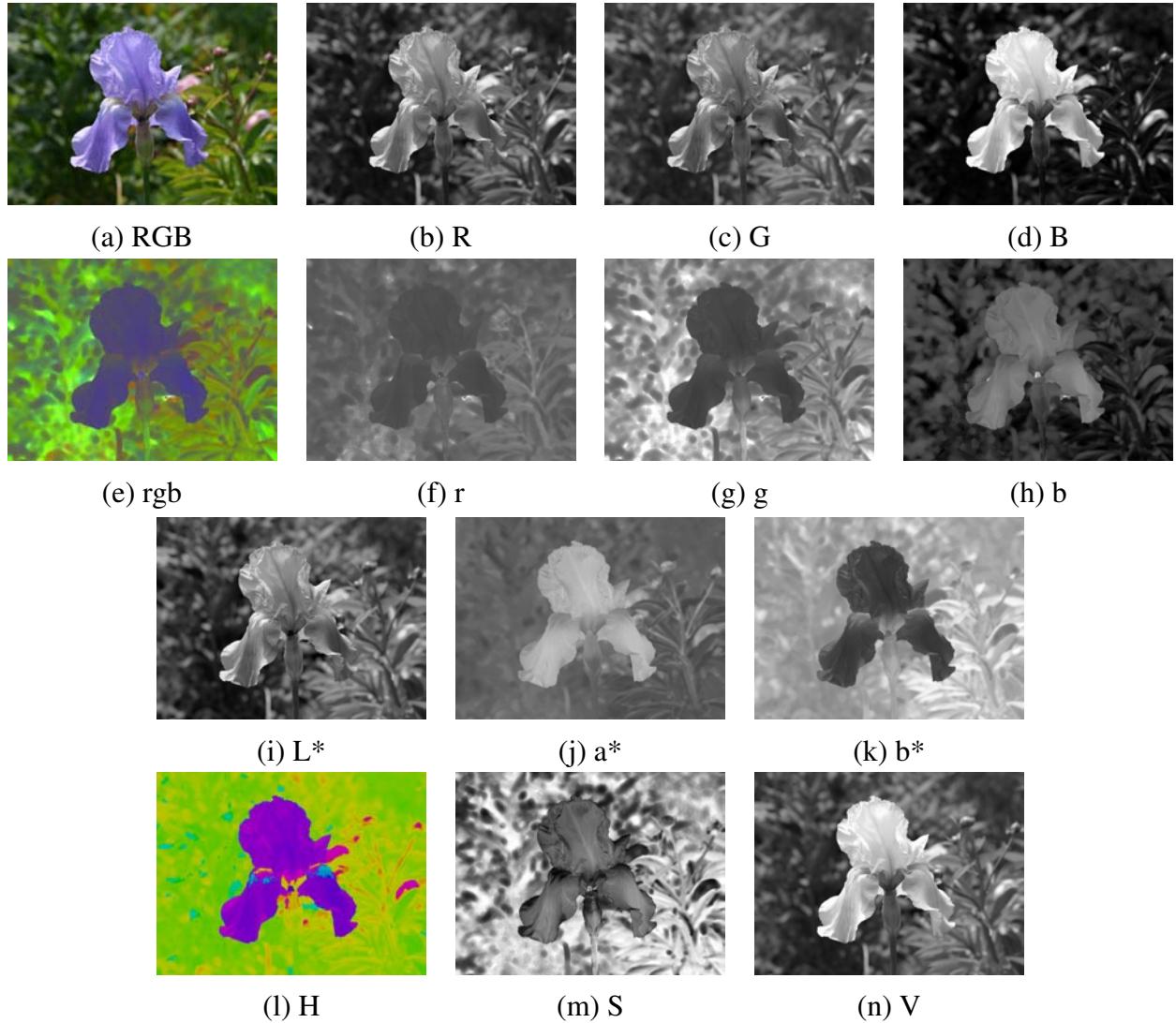


Figure 2.34: *Color space transformations: (a-d) RGB; (e-h) rgb. (i-k) L*a*b*; (l-n) HSV. Note that the rgb, L*a*b*, and HSV values are all re-scaled to fit the dynamic range of the printed page.*

gamma compressed R'G'B' values. However, if you are trying to do careful image de-blocking (Exercise 3.30), this information may be useful.

Another color space you may come across is *hue, saturation, value* (HSV), which is a projection of the RGB color cube onto a non-linear chroma angle, a radial saturation percentage, and a luminance-inspired value. In more detail, value is defined as either the mean or maximum color value, saturation is defined as scaled distance from the diagonal, and hue is defined as the direction around a color wheel (the exact formulas can be found in (Hall 1989, Foley *et al.* 1995)). Such a decomposition is quite natural in graphics applications such as color picking (it approximates the Munsell chart for color description). Figure 2.34l–n shows an HSV representation of a sample color image, where saturation is encoded using a gray scale (saturated = darker) and hue is depicted as a color.

If you want your computer vision algorithm to only affect the value (luminance) of an image and not its saturation or hue, a simpler solution is to use either the Yxy (luminance + chromaticity) coordinates defined in (2.100), or the even simpler *color ratios*,

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B} \quad (2.112)$$

(Figure 2.34e–h). After manipulating the luma (2.108), e.g., through the process of histogram equalization §3.1.4, you can multiply each color ratio by the ratio of the new to old luma to obtain an adjusted RGB triplet.

While all of these color systems may sound confusing, in the end, it often may not matter that much which one you use. Poynton, in his *Color FAQ*, <http://www.poynton.com/ColorFAQ.html>, notes that the perceptually-motivated L*a*b* system is qualitatively similar to the gamma-compressed R'G'B' system we mostly deal with, since both have a fractional power scaling (which approximate a logarithmic response) between the actual intensity values and the numbers being manipulated. As in all cases, think carefully about what you are trying to accomplish before deciding on a technique to use.¹⁴

2.3.3 Compression

The last stage in a camera's processing pipeline is usually some form of image compression (unless you are using a lossless compression scheme such as camera RAW or PNG).

All color video and image compression algorithms start by converting the signal into YCbCr (or some closely related variant), so that they can compress the luminance signal with higher

¹⁴ If you are at a loss for questions at a conference, you can always ask the speaker why he/she did not use a perceptual color space like L*a*b*. Conversely, if they did use L*a*b*, you can ask if they have any concrete evidence that this works better than regular colors. :-)

fidelity than the chrominance signal. (Recall that the human visual system has poorer frequency response to color than to luminance changes.) In video, it is common to subsample Cb and Cr by a factor of two horizontally, while with still images (JPEG), the subsampling (averaging) occurs both horizontally and vertically.

Once the luminance and chrominance images have been appropriately subsampled and separated into individual images, they are then passed to a *block transform* stage. The most common technique used here is the *discrete cosine transform* (DCT), which is a real-valued variant of the discrete Fourier transform (DFT) §3.3.1. The DCT is a reasonable approximation to the Karhunen-Loëve or eigenvalue decomposition of natural image patches, i.e., the decomposition that simultaneously packs the most energy into the first coefficients and diagonalizes the joint covariance matrix among the pixels (makes transform coefficients statistically independent). Both MPEG and JPEG use 8×8 DCT transforms (Wallace 1991, Le Gall 1991), although newer variants use smaller 4×4 blocks, or alternative transforms such as wavelets (Taubman and Marcellin 2002) and lapped transforms (Malvar 1990, Malvar 1998, Malvar 2000) are now used.

After transform coding, the coefficient values are quantized into a set of small integer values that can be coded using a variable bit length scheme such as a Huffman code or an arithmetic code (Wallace 1991). (The DC (lowest frequency) coefficients are also adaptively predicted from the previous block's DC values.) The step size in the quantization is the main variable controlled by the *quality* setting on the JPEG file.

[Note: Need a diagram here, just to break up the monotony...]

With video, it is also usual to perform block-based *motion compensation*, i.e., to encode the difference between each block and a *predicted* set of pixel values obtained from a shifted block in the previous frame. (The exception is the *motion-JPEG* scheme used in older DV camcorders, which is nothing more than a series of individually JPEG compressed image frames.) While basic MPEG uses 16×16 motion compensation blocks with integer motion values (Le Gall 1991), newer standards use adaptively sized block, sub-pixel motions, and the ability to reference blocks from older frames. In order to recover more gracefully from failures and to allow for random access to the video stream, predicted P frames are interleaved among independently coded I frames. (Bidirectionally B frames are also sometimes used.)

While this is just a high-level sketch of how image compression works, it is useful to understand so that the artifact introduced by such techniques can be compensated for in various computer vision applications.

2.4 Exercises

A note to students: This chapter is relatively light on exercises, since it contains mostly back-

ground material and not that many usable techniques. If you really want to understand multi-view geometry in a thorough way, I encourage you to read and do the exercises in ([Hartley and Zisserman 2004](#)). Similarly, if you want some exercises related to the image formation process, ([Glassner 1995](#)) is full of challenging problems.

Ex 2.1 (Least squares intersection point and line fitting (advanced)) Equation (2.4) shows how the intersection of two 2D lines can be expressed as their cross-product, assuming the lines are expressed as homogeneous coordinates.

1. If you are given more than two lines and want to find a point \tilde{x} that minimizes the sum of squared distances to each line,

$$D = \sum_i (\tilde{x} \cdot \tilde{l}_i)^2, \quad (2.113)$$

how can you compute this quantity? (Hint: write the dot product as $\tilde{x}^T \tilde{l}_i$ and turn the squared quantity into a *quadratic form*, $\tilde{x}^T A \tilde{x}$.)

2. To fit a line to a bunch of points, you can compute the *centroid* (mean) of the points as well as the *covariance matrix* of the points around this mean. Show that the line passing through the centroid along the major axis of the covariance ellipsoid (largest eigenvector) minimizes the sum of squared distances to the points.
3. These two approaches are fundamentally different, even though projective duality tells us that points and lines are interchangeable. Why are these two algorithms so apparently different? Are they actually minimizing different objectives?

Ex 2.2 (2D transform editor) Write a program that lets you interactively create a set of rectangles, and then interactively modify their “pose” (2D transform). You should implement the following steps:

1. Open up an empty window (“canvas”).
2. Shift drag (rubber-band) to create a new rectangle.
3. Select the deformation mode (motion model): translation, rigid, similarity, affine, or perspective.
4. Drag any corner of the outline to change its transformation.

This exercise should be built on a set of pixel coordinate and transformation classes, either implemented by yourself, or from some other software library. Persistence of the created representation (save and load) should also be supported (for each rectangle, save its transformation).

Ex 2.3 (3D viewer) Write a simple viewer for 3D points, lines, and polygons. Import a set of point and line commands (primitives) as well as a viewing transform. Interactively modify the object and/or camera transform.

This viewer can be an extension of the one you created in (Ex 2.2). Simply replace the viewing transformations with their 3D equivalents.

Optional: add a z-buffer to do hidden surface removal for polygons.

Optional: use a 3D drawing package instead, but just write the viewer control.

Ex 2.4 (Focus distance and depth of field) Figure out how the focus distance and depth of field indicators on a lens are determined.

1. Compute and plot the focus distance z_o as a function of the distance travelled from the focal length $\Delta z_i = f - z_i$ for a lens of focal length f (say 100mm). Does this explain the hyperbolic progression of focus distances you see on a typical lens (Figure 2.21)?
2. Compute the depth of field (minimum and maximum focus distances) for a given focus setting z_o as a function of the circle of confusion diameter c (make it a fraction of the sensor width), the focal length f , and the f-stop number N (which relates to the aperture diameter d). Does this explain the usual depth of field markings on a lens that bracket the in focus marker, as in Figure 2.21a?
3. Now consider a zoom lens with a varying focal length f . Assume that as you zoom, the lens stays in focus, i.e., the distance from the rear nodal point to the sensor plane z_i adjusts itself automatically for a fixed focus distance z_o . How do the depth of field indicators vary as a function of focal length? Can you reproduce a two-dimensional plot that mimics the curved depth of field lines seen on the lens in Figure 2.21b?

Ex 2.5 (F-numbers and shutter speeds) List the common f-numbers and shutter speeds that your camera provides. On older model SLRs, these are visible on the lens and shutter speed dials. On newer camera, you will have to look at the electronic viewfinder (or LCD screen/indicator) as you manually adjust exposures.

1. Do these form geometric progressions, and if so, with what ratios? How do these relate to exposure values (EVs)?
2. If your camera has shutter speeds of $\frac{1}{60}$ and $\frac{1}{125}$, do you think that these two speeds are exactly a factor of two apart, or a factor of $125/60 = 2.083$ apart?
3. How accurate do you think these numbers are? Can you devise some way to measure exactly how the aperture affects how much light reaches the sensor, and what the exact exposure times actually are?

Ex 2.6 (Noise level calibration) Estimate the amount of noise in your camera by taking repeated shots of a scene with the camera mounted on a tripod. (Purchasing a remote shutter release is a good investment if you own a DSLR.) Alternatively, take a scene with constant color regions (such as a color checker chart) and estimate the variance by fitting a smooth function to each color region and then taking differences from the predicted function.

- Plot your estimated variance as a function of level for each of your color channels separately.
- Change the ISO setting on your camera, or if you cannot do that, reduce the overall light in your scene (turn off lights, draw the curtains, wait until dusk, ...) Does the amount of noise vary a lot with ISO/gain?
- Compare your camera to another one at a different price point or year of make. Is there evidence to suggest that “you get what you pay for”? Does the quality of digital cameras seem to be improving over time?

Ex 2.7 (Gamma correction in image stitching) Here’s a relatively simple puzzler. Assume you are given two images that are part of a panorama that you will be stitching (§8). The two images were taken with different exposures, so you want to adjust the RGB values so that they match along the seam line. Is it necessary to undo the gamma in the color values in order to achieve this?

Ex 2.8 (White point balancing (tricky)) A common (in-camera or post-processing) technique for performing white point adjustment is to take a picture of a white piece of paper and to adjust the RGB values of an image to make this a neutral color.

1. Describe how you would adjust the RGB values in an image given a sample “white color” of (R_w, G_w, B_w) to make this color neutral (without changing the exposure too much).
2. Does your transformation involve a simple (per-channel) scaling of the RGB values, or do you need a full 3×3 color twist matrix (or something else)?
3. Convert your RGB values to XYZ. Does the appropriate correction now only depend on the XY (or xy) values? If so, when you convert back to RGB space, do you need a full 3×3 color twist matrix to achieve the same effect?
4. If you used pure diagonal scaling in the direct RGB mode, but end up with a twist if you work in XYZ space, how do you explain this apparent dichotomy? Which approach is correct? (Or, is it possible that neither approach is actually correct?)
5. If you want to find out what your camera *actually* does, continue on to the next exercise.

Ex 2.9 (Skin color detection) [Note: Move this to later in the book?] Devise a simple skin color detector based on chromaticity. [Note: Need to find a good reference to the fact that chromaticity is a good skin feature. Perhaps ([Forsyth and Fleck 1999](#)) or ([Jones and Rehg 2001](#))? The former uses color opponency in log domain, while the latter uses a large RGB histogram with manually labeled pixels. The students who did the projects on image quality cite ([Vezhnevets et al. 2003](#)). A newer citation forward chaining through Google Scholar is ([Kakumanu et al. 2007](#)).]

1. Take a variety of photographs of people, and calculate the xy chromaticity values for each pixel.
2. Crop the photos, or otherwise indicate with a painting tool which pixels are likely to be skin (face, arms, etc.).
3. Calculate a color (chromaticity) distribution for these pixels. You can use something as simple as a mean and covariance measure, or as complicated as a mean-shift segmentation algorithms §4.5.2. You can optionally use non-skin pixels to model the *background distribution*.
4. Use your computed distribution to find the skin regions in an image. One easy way to visualize this is to paint all non-skin pixels a given color, such as white or black.
5. How sensitive is your algorithm to color balance (scene lighting)?
6. Does a simpler chromaticity measurement such as a color ratio (2.112) work just as well?

Ex 2.10 (In-camera color processing (challenging)) If your camera supports a RAW pixel mode, take a pair of RAW and JPEG images, and see if you can infer what the camera is doing when it converts the RAW pixel values to the final color-corrected and gamma-compressed 8-bit JPEG pixel values.

- Deduce the pattern in your color filter array from the correspondence between co-located RAW and color-mapped pixel values. Use a color checker chart at this stage if it makes your life easier. You may find it helpful to split the RAW image into four separate images (subsampling even/odd columns/rows) and to treat each of these new images as a “virtual” sensor.
- Evaluate the quality of the de-mosaicing algorithm by taking pictures of challenging scenes which contain strong color edges (Figure 9.32 in §9.3.1).
- If you can take the same exact picture after changing the color balance values in your camera, compare how these setting affect this processing.

Chapter 3

Image processing

3.1	Local operators	99
3.1.1	Pixel transforms	101
3.1.2	Color transforms	102
3.1.3	Compositing and matting	103
3.1.4	Histogram equalization	105
3.1.5	<i>Application:</i> Tonal adjustment	109
3.2	Neighborhood operators	109
3.2.1	Linear filtering	111
3.2.2	Non-linear filtering	121
3.2.3	Morphology	126
3.2.4	Distance transforms	127
3.2.5	Connected components	129
3.3	Fourier transforms	131
3.3.1	Wiener filtering	139
3.3.2	<i>Application:</i> Sharpening, blur, and noise removal	143
3.4	Pyramids and wavelets	144
3.4.1	Interpolation and decimation	145
3.4.2	Multi-resolution representations	151
3.4.3	Wavelets	156
3.4.4	<i>Application:</i> Image blending	162
3.5	Geometric transformations	164
3.5.1	Parametric transformations	165
3.5.2	Mesh-based warping	171
3.5.3	<i>Application:</i> Feature-based morphing	174
3.6	Global optimization	175
3.6.1	Regularization	176
3.6.2	Markov Random Fields	181
3.6.3	<i>Application:</i> Image restoration	194
3.7	Exercises	194



Figure 3.1: Some common image processing operations: (a) original image; (b) increased contrast; (c) change in hue; (d) “posterized” (quantized colors); (e) blurred; (f) rotated.

Now that we have seen how images are formed through the interaction of 3D scene elements, lighting, and camera optics and sensors, let us look at the first stage of most computer vision applications, namely the use of image processing to preprocess the image and convert it into a form suitable for further analysis. Examples of such operations include the exposure correction and color balancing, the reduction of image noise, an increase in sharpness, or straightening the image by rotating it (Figure 3.1).

In this chapter, I review standard *image processing* operators that map pixel values from one image to another. Image processing is often taught in electrical engineering departments as a follow-on course to a more introductory course in *signal processing* (Oppenheim and Schafer 1996, Oppenheim *et al.* 1999). Popular textbooks for image processing include (Lim 1990, Crane 1997, Gomes and Velho 1997, Jähne 1997, Pratt 2001, Gonzales and Woods 2002, Russ 2007).

I begin this chapter with the simplest kind of image transforms, namely those that manipulate each pixel independently of its neighbors (§3.1). Such transforms are often called *local operators* or *point processes*. Next, I examine *neighborhood* (area-based) operators, where each new pixel's value depends on a small number of neighboring input pixel values (§3.2). A convenient tool to analyze (and sometimes accelerate) such neighborhood operations in the *Fourier Transform*, which I cover in §3.3. Neighborhood operators can be cascaded to form *image pyramids* and *wavelets*, which are useful for analyzing images at a variety of resolutions (scales) and for accelerating certain operations (§3.4). Another important class of global operators are *geometric transformations* such as rotations, shears, and perspective deformations (§3.5). Finally, I introduce *global optimization* approaches to image processing, which involve the minimization of an energy functional, or equivalently, optimal estimation using Bayesian *Markov Random Field* models (§3.6).

3.1 Local operators

The simplest kinds of image processing transforms are *local operators*, where each output pixel's value only depends on the corresponding input pixel value (plus, potentially, some globally collected information or parameters). Examples of such operators include brightness and contrast adjustments (Figure 3.2) as well as color correction and transformations. In the image processing literature, such operations are also known as *point processes* (Crane 1997).

I begin this section with a quick review of simple local operators such as brightness scaling and image addition. Next, I discuss how colors in images can be manipulated. I then present *image compositing* and *matting* operations, which play an important role in computational photography (§9) and computer graphics applications. Finally, I describe the more global process of *histogram equalization*. I close with an example application that manipulates *tonal values* (exposure and contrast) to improve image appearance.

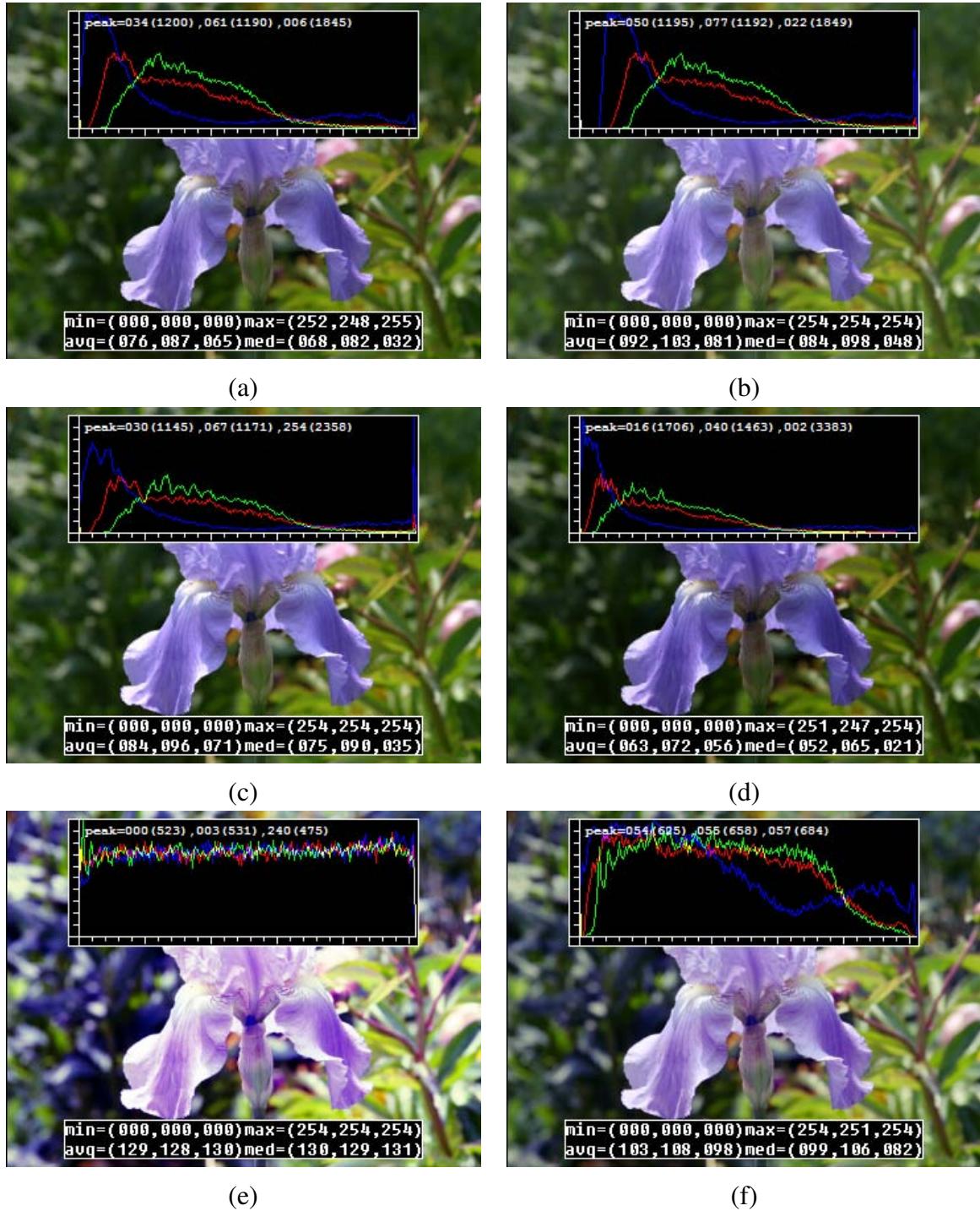


Figure 3.2: Some local image processing operations: (a) original image along with its three color (per-channel) histograms; (b) brightness increased (additive offset, $b = 16$); (c) contrast increased (multiplicative gain $a = 1.1$); (d) gamma (partially) linearized ($\gamma = 1.2$); (e) full histogram equalization; (f) partial histogram equalization.

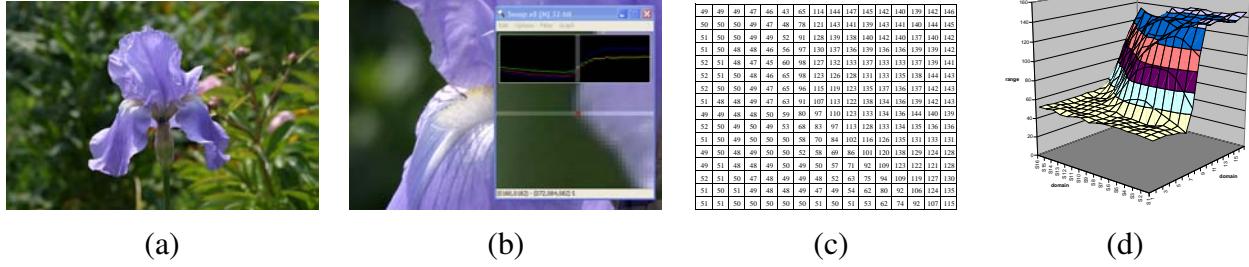


Figure 3.3: Visualizing image data: (a) original image; (b) cropped portion and scanline plot using an image inspection tool; (c) grid of numbers; (d) surface plot. For figures (c)–(d), the image was first converted to grayscale.

3.1.1 Pixel transforms

An image processing *operator* is a function that takes one or more input images (signals) and produces an output image. In the continuous domain, this can be denoted as

$$g(\mathbf{x}) = h(f(\mathbf{x})) \text{ or } g(\mathbf{x}) = h(f_0(\mathbf{x}), \dots, f_n(\mathbf{x})), \quad (3.1)$$

where \mathbf{x} is the D-dimensional *domain* of the functions (usually $D = 2$ for images), and the functions f and g operate over some *range*, which can either be scalar or vector-valued (e.g., for color images or 2D motion). For discrete (sampled) images, the domain consists of a finite number of *pixel locations*, $\mathbf{x} = (i, j)$, and we can write

$$g(i, j) = h(f(i, j)). \quad (3.2)$$

Figure 3.3 shows how an image can be represented either by its color (appearance), as a grid of numbers, or as a two-dimensional function (surface plot).

Two commonly used point processes are multiplication and addition with a constant,

$$g(\mathbf{x}) = af(\mathbf{x}) + b. \quad (3.3)$$

The parameters $a > 0$ and b are often called the *gain* and *bias* parameters; sometimes these parameters are said to control *contrast* and *brightness*, respectively (Figures 3.2 b–c). [Note: Look in (Kopf et al. 2007b) and see if key and something else is used as well.] The bias and gain parameters can also be spatially varying,

$$g(\mathbf{x}) = a(\mathbf{x})f(\mathbf{x}) + b(\mathbf{x}), \quad (3.4)$$

e.g., when simulating the *graded density filter* used by photographers to selectively darken the sky.

Multiplicative gain (both global and spatially varying) is a *linear* operation, since it obeys the *superposition principle*,

$$h(f_0 + f_1) = h(f_0) + h(f_1). \quad (3.5)$$

(I will have more to say about linear shift invariant operators in §3.2.1.) Operators such as image squaring (which is often used to get a local estimate of the *energy* in a band-pass filtered signal §3.4) are not linear.

Another commonly used *diadic* (two input) operator is the *linear blend* operator,

$$g(\mathbf{x}) = (1 - \alpha)f_0(\mathbf{x}) + \alpha f_1(\mathbf{x}). \quad (3.6)$$

By varying α from $0 \rightarrow 1$, this operator can be used to perform a temporal *cross-dissolve* between two images or videos, as seen in slide shows and film production, or as a component of image *morphing* algorithms (§3.5.3).

One highly used non-linear transform that is often applied to images before further processing is *gamma correction*, which is used to remove the non-linear mapping between input radiance and quantized pixel values (§2.3.2). To invert the gamma mapping applied by the sensor, we can use

$$g(\mathbf{x}) = [f(\mathbf{x})]^{1/\gamma}, \quad (3.7)$$

where a gamma value of $\gamma \approx 2.2$ is a reasonable fit for most digital cameras.

3.1.2 Color transforms

While color images can be treated as arbitrary vector-valued functions or collections of independent bands, it usually makes sense to think about them as highly correlated signals with strong connections to the image formation process (§2.2), sensor design (§2.3), and human perception (§2.3.2). Consider, for example, brightening a picture by adding a constant value to all three channels, as shown in Figure 3.2b. Can you tell if this achieves the desired effect of making the image look brighter? Can you see any undesirable side-effects or artifacts?

In fact, adding the same value to each color channel not only increases the apparent *intensity* of each pixel, it can also affect the pixel's *hue* and *saturation*. How can we define and manipulate such quantities in order to achieve the desired perceptual effects?

As discussed in §2.3.2, chromaticity coordinates (2.100) or even simpler color ratios (2.112) can first be computed and then used after manipulating (e.g., brightening) the luminance Y to re-compute a valid RGB image with the same hue and saturation. Figure 2.34g–i in the previous chapter shows some color ratio images, multiplied by the middle gray value for better visualization.

Similarly, color balancing (e.g., to compensate for incandescent lighting) can be performed by either multiplying each channel with a different scale factor, or by the more complex process of mapping to XYZ color space, changing the nominal whitepoint, and mapping back to RGB, which can be written down using a linear 3×3 *color twist* transform matrix. As mentioned in §2.3.2, Exercise 2.8 has you explore some of these issues, as does Exercise 3.1.



Figure 3.4: *Image matting and compositing* (Chuang et al. 2001): (a) source image; (b) extracted foreground object F ; (c) alpha matte α shown in grayscale; (d) new composite C .

$$\begin{array}{ccccc}
 \boxed{\text{Note: } \text{RGBA}_{\text{back}}} & \times & \boxed{(1 - \text{Note: } A_{\text{front}})} & + & \boxed{\text{Note: } \text{RGBA}_{\text{front}}} \\
 B & & \alpha & & \alpha F \\
 \text{(a)} & & \text{(b)} & & \text{(c)} \\
 & & & & = \\
 & & & & \boxed{\text{Note: } \text{RGBA}_{\text{comp}}} \\
 & & & & \text{C} \\
 & & & & \text{(d)}
 \end{array}$$

Figure 3.5: *Compositing equation* $C = (1 - \alpha)B + \alpha F$.

[Note: These images, shown as pixel grids with numbers inside, could be close-ups from the previous image. If so, draw a square in the previous figure and comment on it here. Note that an earlier version of this figure and text used the notation $\text{RGBA}_{\text{front}}$ instead of F , etc.]

Another fun project, best attempted after you have mastered the rest of the material in this chapter, is to take a picture with a rainbow in it (Figure 3.7) and enhance the strength of the rainbow.

3.1.3 Compositing and matting

In many photo editing and visual effects applications, it is often desirable to cut a *foreground* object out of one scene and put it on top of a different *background* (Figure 3.4). The process of extracting the object from the original image is often called *matting* (Smith and Blinn 1996), while the process of inserting it into another image (without visible artifacts) is called *compositing* (Porter and Duff 1984, Blinn 1994a).

The intermediate representation used for the foreground object between these two stages is called an *alpha-matted color image* (Figure 3.4b–c). In addition to the three color RGB channels, an alpha-matted image contains a fourth *alpha* channel α (or A) that describes the relative amount of *opacity* or *fractional coverage* at each pixel (Figures 3.4c and 3.5b). The opacity is the inverse of the *transparency*. Pixels within the object are fully opaque ($\alpha = 1$), while pixels fully outside



Figure 3.6: Example of light reflecting off the transparent glass of a picture frame (Szeliski et al. 2000). You can clearly see the woman's portrait inside the picture frame superimposed with the reflection of a man's face off the glass.

of the object are transparent ($\alpha = 0$). Pixels on the boundary of the object vary smoothly between these two extremes, which hides the perceptual visible *jaggies* that occur if only binary opacities are used.

To composite a new (or foreground) image on top of an old (background) image, the *over operator* (first proposed by Porter and Duff (1984) and then studied extensively by Blinn (1994a)(1994b)) is used,

$$C = (1 - \alpha)B + \alpha F. \quad (3.8)$$

This operator *attenuates* the influence of the background image B by a factor $(1 - \alpha)$, and then adds in the color (and opacity) values corresponding to the foreground layer F .

In many situations, it is convenient to represent the foreground colors in *pre-multiplied* form, i.e., to store (and manipulate) the αF values directly. As Blinn (1994b) shows, the pre-multiplied RGBA representation is preferred for several reasons, including the ability to blur or resample (e.g., rotate) alpha-matted images without any additional complications (just treating each RGBA band independently). However, when matting using local color consistency (Ruzon and Tomasi 2000, Chuang *et al.* 2001), the pure un-multiplied foreground colors F are used, since these remain constant (or vary slowly) in the vicinity of the object edge.

The over operation is not the only kind of compositing operation that can be used. Porter and Duff (1984) describe a number of additional operations that can be useful in photo editing and visual effects applications. In this book, we only concern ourselves with one additional, commonly occurring case (but see Exercise 3.2).

When light reflects off of clean transparent glass, the light passing through the glass and the light reflecting off the glass are simply added together (Figure 3.6). This model is useful in the analysis of *transparent motion* (Black and Anandan 1996, Szeliski *et al.* 2000), which occurs when

such scenes are observed from a moving camera (§7.5).

The actual process of *matting*, i.e., recovering the foreground, background, and alpha matte values from one or more images, has a rich history, which we will study in §9.4. Smith and Blinn (1996) have a nice survey of traditional *blue-screen matting* techniques, while Toyama *et al.* (1999) review *difference matting*. More recently, there has been a lot of activity in computational photography relating to *natural image matting* (Ruzon and Tomasi 2000, Chuang *et al.* 2001), which attempts to extract the mattes from a single natural image (Figure 3.4a) or from extended video sequences (Chuang *et al.* 2002). All of these techniques are described in more detail in §9.4.

3.1.4 Histogram equalization

While the brightness and gain controls described in §3.1.1 can improve the appearance of an image, how can we automatically determine their best values? One approach might be to look at the darkest and brightest pixel values in an image, and to map these to pure black and pure white. Another approach might be to find the *average* value in the image and push this towards middle gray and to expand the *range* so that it more closely fill the displayable values (Kopf *et al.* 2007b).

How can we visualize the set of lightness values in an image in order to test some of these heuristics? The answer is to plot the *histogram* of the image lightness values, as shown in Figure 3.7b.¹ [Note: For this Figure, I have converted a color image into its equivalent grayscale intensity, using the color to grayscale formula (2.108) presented in §3.1.2. Alternatively, consider keeping color images, and just displaying the grayscale histograms. Find out what my camera and Photo Gallery do when displaying histograms. They both show some kind of grayscale (luminance?) histogram.] From this distribution, we can compute relevant statistics such as the minimum, maximum, and average intensity values. Notice, however, that the image has both an excess of dark values and light values, but that the mid-range values are largely underpopulated. [Note: Need to find such an image, should not be hard...] Would it not be better if we could simultaneously brighten some dark values and darken some light values, while still using the full extent of the available dynamic range? Can you think of a mapping that might do this?

One popular answer to this question is to perform *histogram equalization*, i.e., to find an intensity mapping function $f(I)$ such that the resulting histogram is flat. The trick to finding such a mapping is the same one that people use to generate random samples from a *probability density function*, which is to first compute the *cumulative distribution function* shown in Figure 3.7c.

Think of the original histogram $h(I)$ as the distribution of grades in a class after some exam. How can we map a particular grade to its corresponding *percentile*, so that the students at the

¹ The histogram is simply the *count* of the number of pixels at each gray level value. For an 8-bit image, an accumulation table with 256 entries is needed. For higher bit depths, a table with the appropriate number of entries (probably fewer than the full number of gray levels) should be used.

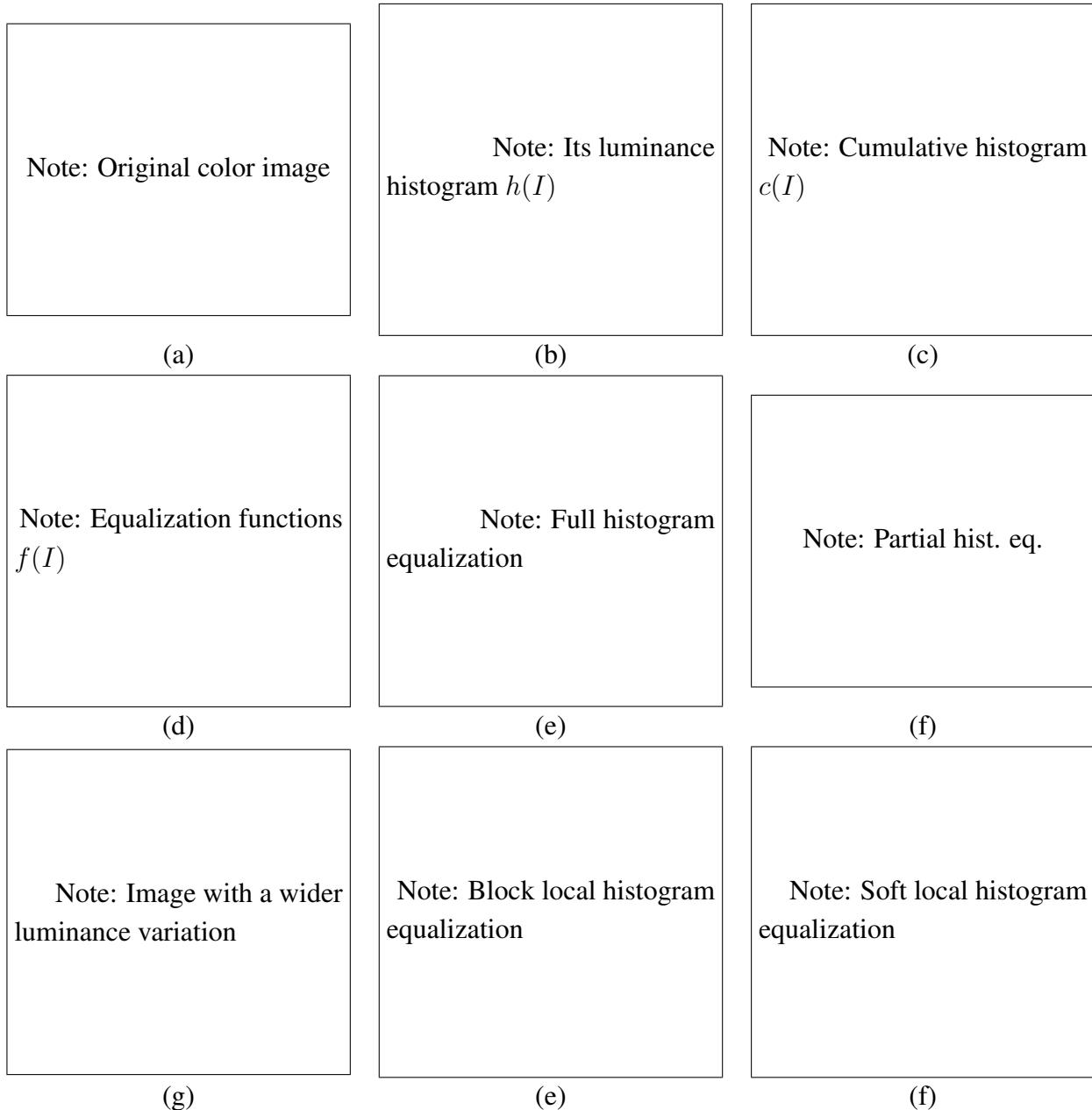


Figure 3.7: *Histogram analysis and equalization:* (a) original image (b) intensity histogram; (c) cumulative distribution function; (d) equalization (transfer) functions; (e) full histogram equalization; (f) partial histogram equalization; (g) another sample image; (h) block histogram equalization; (i) locally adaptive histogram equalization.

[Note: Generate these images.]

75% percentile range scored better than $3/4$ of their classmates? The answer is to integrate the distribution $h(I)$ to obtain the cumulative distribution $c(I)$,

$$c(I) = \frac{1}{N} \sum_{i=0}^I h(i) = c(I-1) + \frac{1}{N} h(I), \quad (3.9)$$

where N is the number of pixels in the image (oops! I mean students in the class :-). For any given grade/intensity, we can look up its corresponding percentile $c(I)$ and determine the final value that pixel should take. (When working with 8-bit pixel values, the I and c axes are rescaled to go from $[0, 255]$.)

Figure 3.7d shows the result of applying $f(I) = c(I)$ to the original image. As we can see, the resulting histogram is flat, but so is the resulting image (“flat” in the sense of lack of contrast and muddy looking). One way to compensate for this is to only *partially* compensate for the histogram unevenness, e.g., by using a mapping function $f(I) = \alpha c(I) + (1-\alpha)I$, i.e., a linear blend between the cumulative distribution function and the identity transform (straight line). As you can see in Figure 3.7e, the resulting image maintains more of its original grayscale distribution while having a more appealing balance.

Another potential problem with histogram equalization (or in general, image brightening) is that noise in dark regions can be amplified and become more visible. [Note: Need a figure here] The exercise on histogram equalization Exercise 3.6 suggests some possible ways to mitigate this, as well as alternative techniques to maintain contrast and “punch” in the original images. [Note: Larson et al. (1997) develop a technique that works in the log brightness domain and doesn’t allow the “gain” to exceed a unit amount.] [Note: Noise boosting becomes even more visible in local adaptation (next paragraph): defer this discussion until after that?]

Local histogram equalization

While global histogram equalization can be useful, for some images, it might be preferable to apply different kinds of equalization in different regions. Consider for example the image in Figure 3.7g, which has a wide range of luminance values. Instead of computing a single curve ([Note: show result for global?]), what if we were to subdivide the image into $M \times M$ pixel blocks and perform separate histogram equalization in each sub-block? As you can see in Figure 3.7h, the resulting image is far from ideal, exhibiting a strong patchwork of blocking artifacts (intensity discontinuities at block boundaries).

One way to eliminate blocking artifacts is to use a *moving window*, i.e., to recompute the histogram for every $M \times M$ block centered at each pixel. This process can be quite slow (M^2 operations per pixel), although with clever programming, only the histogram entries corresponding to the pixels entering and leaving the block (in a raster scan across the image) need to be updated

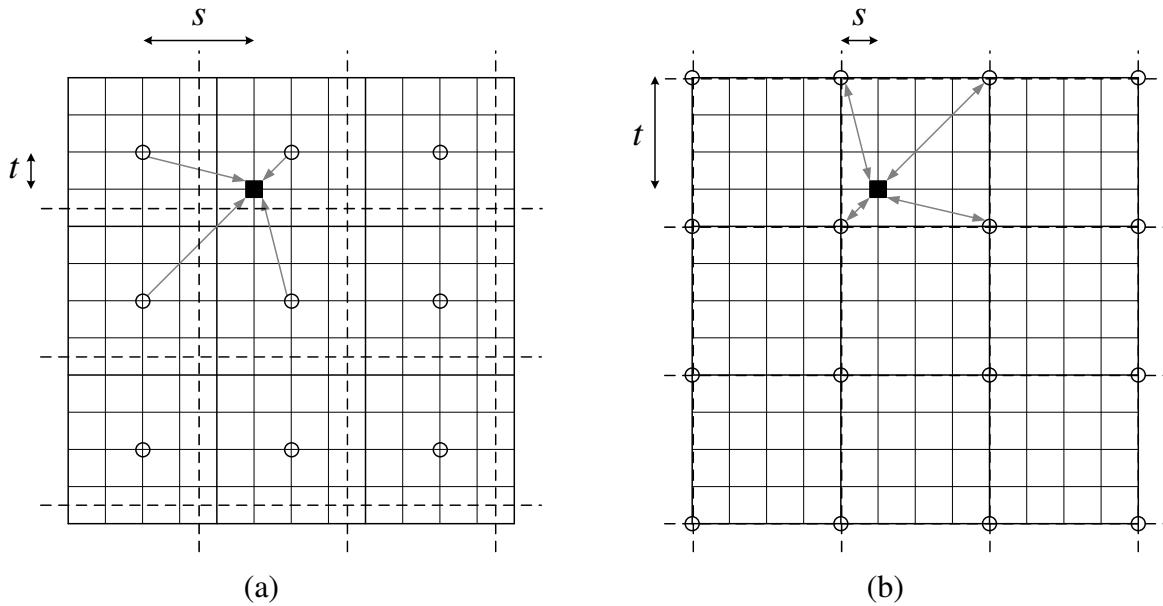


Figure 3.8: *Local histogram interpolation using relative (s, t) coordinates:* (a) *block-based histograms*, with *block centers* shown as *circles*; (b) *corner-based ‘spline’ histograms*. *Pixels are located on grid intersections. The black square pixel’s transfer function is interpolated from the four adjacent lookup tables (gray arrows) using the computed (s, t) values. Block boundaries are shown as dashed lines.*

(M operations per pixel). Note that this operation is an example of the *non-linear neighborhood operations* we will study in more detail in §3.2.2. [Note: Re-read Cox et al. (1995) histogram equalization for stereo and see if it’s this kind.]

A more efficient approach is to compute non-overlapped block-based equalization functions as before, but to then smoothly interpolate the transfer functions as we move between blocks. The weighting function for a given pixel (i, j) can be computed as a function of its horizontal and vertical position (s, t) within a block, as shown in Figure 3.8a. To blend the four lookup functions $\{f_{00}, \dots, f_{11}\}$, a *bilinear* blending function,

$$f_{s,t}(I) = (1-s)(1-t)f_{00}(I) + s(1-t)f_{10}(I) + (1-s)tf_{01}(I) + stf_{11}(I) \quad (3.10)$$

can be used. (See §3.4.1 for higher-order generalizations of such *spline* functions.) Note that instead of blending the four lookup tables for each output pixel (which would be quite slow), it is equivalent to blend the results of mapping a given pixel through the four neighboring lookups. [Note: is there a reference to my work? TR version? Nope. I could cite the patent, but don’t bother, unless in the process of re-implementing this, I come across some new ideas.]

A variant on this algorithm is to place the lookup tables at the *corners* of each $M \times M$ block

(see Figure 3.8b and Exercise 3.7). In addition to blending four lookups to compute the final value, we can also *distribute* each input pixel into four adjacent lookup tables during the histogram accumulation phase (notice that the gray arrows in Figure 3.8b point both ways), i.e.,

$$h_{k,l}(I(i,j)) += w(i,j,k,l), \quad (3.11)$$

where $w(i,j,k,l)$ is the bilinear weighting function between pixel (i,j) and lookup table (k,l) . This is an example of *soft histogramming*, which is used in a variety of other applications, including the construction of SIFT feature descriptors (§4.1.3) and vocabulary trees (§14.3.2).

[Note: Look at the [Chen et al. \(2007\)](#) paper, which has local histogram equalization, and also cites Matlab's *adaphisteq*, which I should try to get source code to and see how it works.]

3.1.5 Application: Tonal adjustment

[Note: Still need to write this section.]

Test out a variety of photo enhancement / effects filters: basic black-point / white-point setting, histogram equalization, color compensation.

Try some standard packages (Photoshop, Picassa, Windows Live Photo Gallery, Office Picture Manager) and try to “black-box” what they do.

Tie this into an exercise, and have students put together a bunch of algorithms and try to “guess” the best compensation parameters for some images.

3.2 Neighborhood operators

Locally adaptive histogram equalization is an example of a *neighborhood operator*, which uses a collection of pixel values in the vicinity of a given pixel to determine its final output value (Figure 3.10a). In addition to performing local tone adjustment, neighborhood operators can be used to *filter* images in order to remove noise, sharpen details, accentuate edges, or add soft blur (Figure 3.9b–d). In this section, I describe both linear and non-linear neighborhood operators, which include as a special case *morphological* operators that operate on binary images. I also describe *semi-global* operators that compute *distance transforms* and find *connected components* in binary images (Figure 3.9f–h).

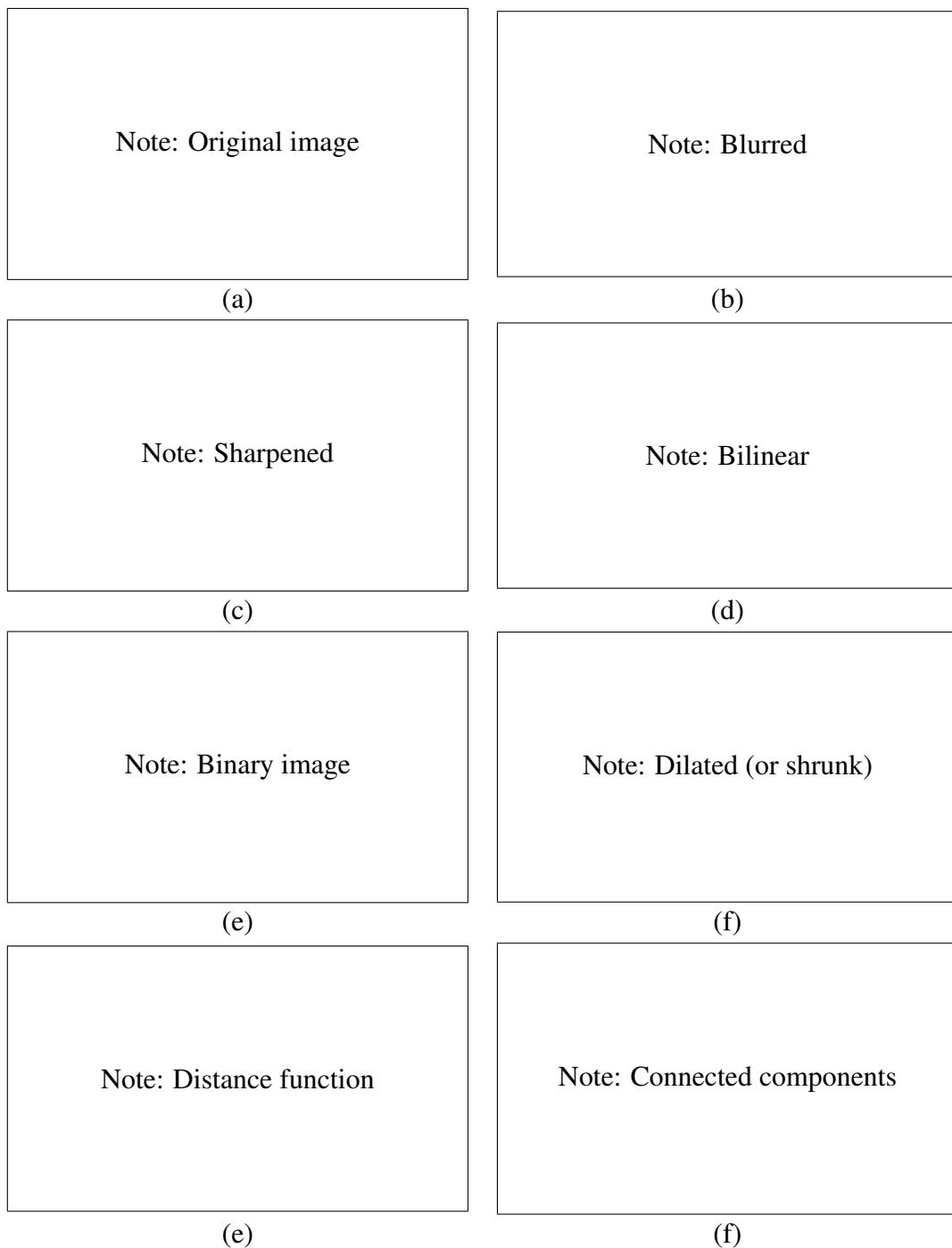


Figure 3.9: *Some neighborhood operations: (a) original image; (b) blurred; (c) sharpened; (d) smoothed with edge preserving filter; (e) binary image; (f) dilated; (g) distance transform; (h) connected components.*

[Note: Need to generate these figures...]

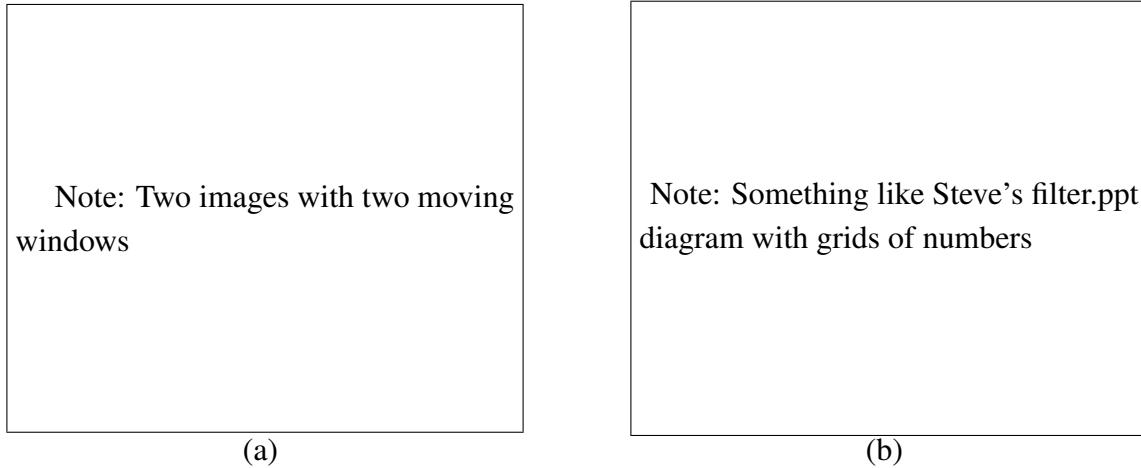


Figure 3.10: *Neighborhood filtering*: (a) each output pixel depends on some function of its local neighborhood; (b) example of pixel transformation (blur).

[Note: Need to generate these figures...]

3.2.1 Linear filtering

The most commonly used type of neighborhood operator is a *linear filter*, in which an output pixel's value is determined as a weighted sum of input pixel values,

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l) \quad (3.12)$$

(Figure 3.10b). The entries in the weight *kernel* or *mask* $h(k, l)$ are often called the *filter coefficients*. The correlation operator can be more compactly notated as

$$g = f \otimes h. \quad (3.13)$$

A common variant on this formula is

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l) = f(k, l)h(i - k, j - l), \quad (3.14)$$

where the sign of the offsets in f has been reversed. This is called the *convolution* operator,

$$g = f * h, \quad (3.15)$$

and h is then called the *impulse response function*.² The reason for this name is that the kernel function, h , convolved with an impulse signal, $\delta(i, j)$ (an image that is 0 everywhere except at the

² The continuous version of convolution can be written as $g(x) = \int f(x - u)h(u)du$.

$$\begin{bmatrix} 72 & 82 & 62 & 52 & 37 \end{bmatrix} * \begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix} \Leftrightarrow \frac{1}{4} \begin{bmatrix} 2 & 1 & . & . & . \\ 1 & 2 & 1 & . & . \\ . & 1 & 2 & 1 & . \\ . & . & 1 & 2 & 1 \\ . & . & . & 1 & 2 \end{bmatrix} \begin{bmatrix} 72 \\ 88 \\ 62 \\ 52 \\ 37 \end{bmatrix}$$

Figure 3.11: One dimensional signal convolution as a sparse matrix-vector multiply, $\mathbf{g} = \mathbf{H}\mathbf{f}$.

origin) reproduces itself, $h * \delta = h$, whereas correlation produces the reflected signal. (Try this yourself to verify that this is so.)

In fact, (3.14) can be interpreted as the superposition (addition) of shifted impulse response functions $h(i - k, j - l)$ multiplied by the input pixel values $f(k, l)$. Convolution has additional nice properties, e.g., it is both commutative and associative. As well, the Fourier transform of two convolved images is the product of their individual Fourier transforms (§3.3).

Both correlation and convolution are *linear shift-invariant* (LSI) operators, which obey both the superposition principle (3.5),

$$h \circ (f_0 + f_1) = h \circ f_0 + h \circ f_1, \quad (3.16)$$

and the *shift invariance* principle,

$$g(i, j) = f(i + k, j + l) \Leftrightarrow (h \circ g)(i, j) = (h \circ f)(i + k, j + l), \quad (3.17)$$

which means that shifting a signal commutes with applying the operator (\circ stands for the LSI operator). Another way to think of shift invariance is that the operator “behaves the same everywhere”.

Occasionally, a shift-variant version of correlation or convolution may be used, e.g.,

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l; i, j), \quad (3.18)$$

where $h(k, l; i, j)$ is the convolution kernel at pixel (i, j) . (Think for example of modeling the spatially varying blur in an image that is selectively defocused depending on depth.)

Correlation and convolution can both be written as a matrix-vector multiply, if we first convert the two-dimensional images $f(i, j)$ and $g(i, j)$ into some raster-ordered vectors \mathbf{f} and \mathbf{g} ,

$$\mathbf{g} = \mathbf{H}\mathbf{f}, \quad (3.19)$$

where the (sparse) \mathbf{H} matrix contains the convolution kernels. Figure 3.11 shows how a one-dimensional convolution can be represented in matrix-vector form.

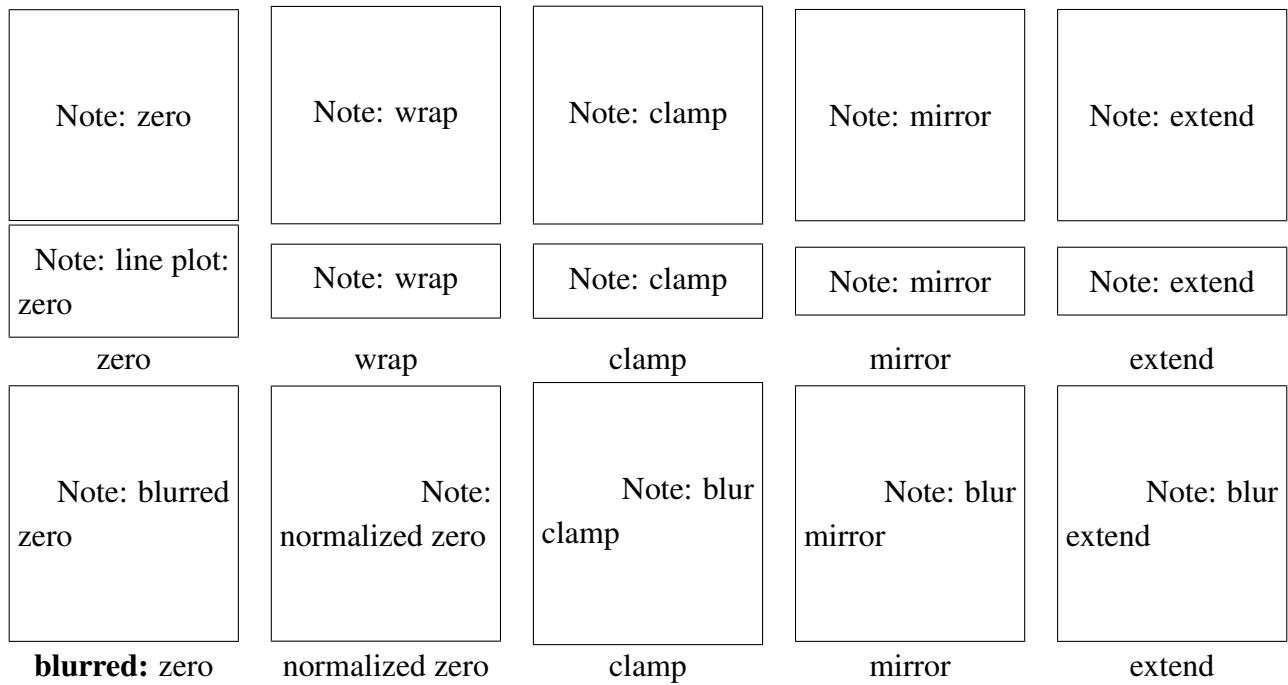


Figure 3.12: *Border padding and the results of blurring the padded image. Top row: padded images; middle row: cross-section; bottom row: blurred padded image. The image below wrapping is the result of dividing (normalizing) the blurred zero-padded RGBA image by its corresponding soft alpha value.*

[Note: Could replace the cross-section plots with an additional figure (Excel?) that show the effects of padding on a smaller synthetic example.]

Padding (border effects)

The astute reader will notice that the matrix multiply shown in Figure 3.11 suffers from *boundary effects*, i.e., the results of filtering the image in this form will lead to a *darkening* of the corner pixels. This is because the original image is effectively being padded with 0 values wherever the convolution kernel extends beyond the original image boundaries.

To compensate for this, a number of alternative *padding* or extension modes have been developed (Figure 3.12):

1. *zero pad*: set all pixels outside the source image to 0 (good choice for alpha-matted cutout images);
2. *constant pad (aka clamp to border)*: set all pixels outside the source image to a specified *border* value;
3. *clamp (aka replicate or clamp to edge)*: repeat edge pixels indefinitely;

4. (*cyclic*) *wrap* (aka *repeat*, or *tile*): loop “around” the image in a “toroidal” configuration;
5. *mirror*: reflect pixels across the image edge;
6. *extend*: extend the signal by subtracting the mirrored version of the signal from the edge pixel value

In the computer graphics literature (Akenine-Möller and Haines 2002, p. 124), these mechanisms are known as the *wrapping mode* (OpenGL) or *texture addressing mode* (Direct3D). The formulas for each of these modes are left as an exercise in Exercise 3.8.

Figure 3.12 shows the effects of padding an image with each of the above mechanisms, as well as the result of blurring the resulting padded image. As you can see, zero padding darkens the edges, replication padding propagates border values inward, reflection padding preserves colors near the borders, and extension keeps the border pixels fixed (during blur).

An alternative to padding is to blur the zero-padded RGBA image and to then divide the resulting image by its alpha value to remove the darkening effect. The results can be quite good, as seen in the blurred image below cyclic replication in Figure 3.12.³

Separable filtering

The process of performing a convolution requires K^2 (multiply-add) operations per pixel, where K is the size (width or height) of the convolution kernel (e.g., the box filter in Figure 3.13a). In many cases, this operation can be significantly sped up by first performing a one-dimensional horizontal convolution followed by a one-dimensional vertical convolution (which requires a total of $2K$ operations per pixel). A convolution kernel for which this is possible is said to be *separable*.

It is easy to show that the two-dimensional kernel \mathbf{K} corresponding to successive convolution with a horizontal kernel \mathbf{h} and a vertical kernel \mathbf{v} is the *outer product* of the two kernels,

$$\mathbf{K} = \mathbf{v}\mathbf{h}^T \quad (3.20)$$

(see Figure 3.13 for some examples). Because of the increased efficiency, the design of convolution kernels for computer vision applications is often influenced by their separability.

How can we tell if a given kernel \mathbf{K} is indeed separable? This can often be done by inspection, or by looking at the analytic form of the kernel (Freeman and Adelson 1991). A more direct method is to treat the 2D kernel as a 2D matrix \mathbf{K} and to take its singular value decomposition,

$$\mathbf{K} = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^T \quad (3.21)$$

³ One can think of this as a very simple but special version of *bilateral filtering* (§3.2.2).

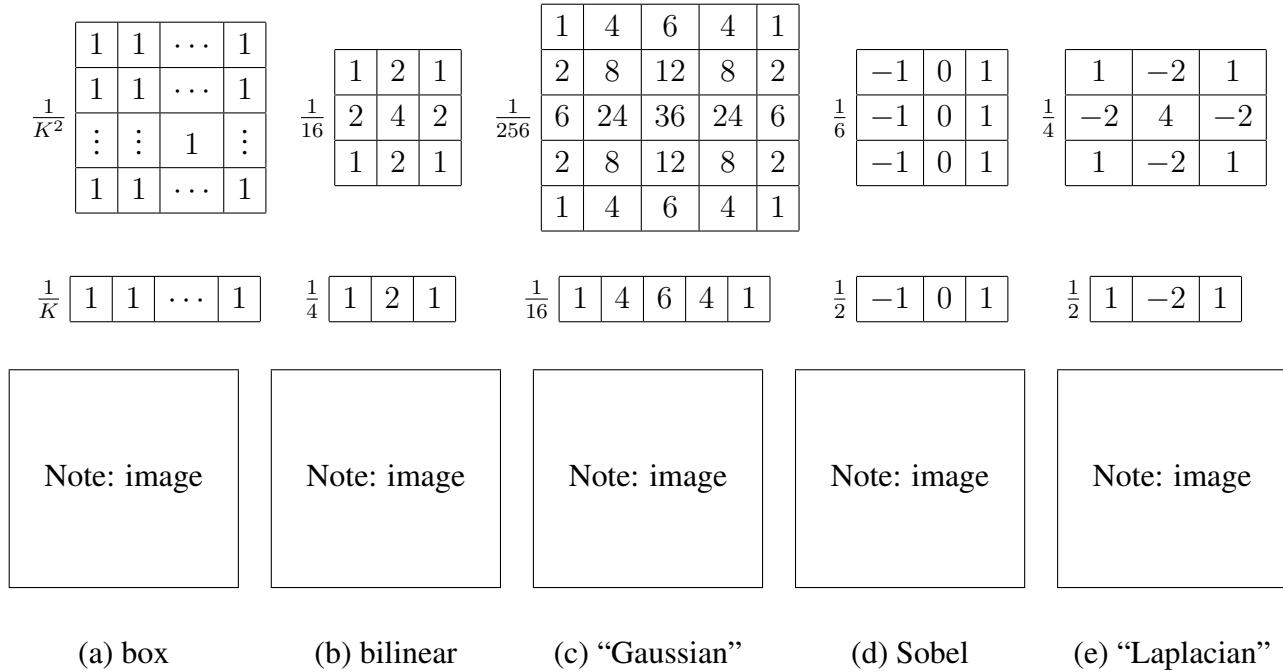


Figure 3.13: *Examples of separable linear filters. Top row: 2D filter kernels; middle row: their corresponding horizontal 1-D kernels; bottom row: filtered images. The filtered Sobel and “Laplacian” images are signed images, and are hence displayed with an added gray offset.*

[Note: For the sample image, can use a synthetic image with a box, a circle, and a triangle, perhaps the Einstein image in the corner.]

(see Appendix A.1 for the definition of the SVD). If only the first singular value σ_0 is non-zero, the kernel is separable and $\sqrt{\sigma_0}u_0$ and $\sqrt{\sigma_0}v_0^T$ provide the vertical and horizontal kernels (Perona 1995).

What if your kernel is not separable, and yet you still want a faster way to implement it? Perona (1995), who first made the link between kernel separability and SVD, suggests using more terms in the (3.21) series, i.e., summing up a number of separable convolutions. Whether this is worth doing or not depends on the relative sizes of K and the number of significant singular values, as well as other considerations such as cache coherency and memory locality.

[Note: Re-read (Perona 1995); also, (Crane 1997) mentions (J. S. Wiejak and Buxton 1985), but that only deals with $\nabla^2 G$.]

Examples of linear filtering

Now that I have described the process for performing linear filtering, let us examine a number of frequently used filters (Figure 3.13).

The simplest filter to implement is the *moving average* or *box* filter, which simply averages the pixel values in a $K \times K$ window. This is equivalent to convolving the image with a kernel of all ones and then scaling (Figure 3.13a). For large kernels, a more efficient implementation is to slide a moving window across each scanline (in a separable filter) while adding the newest pixel and subtracting the oldest pixel from the running sum. (This is related to the concept of *summed area tables*, which I will describe shortly.)

A smoother image can be obtained by separably convolving the image with a piecewise linear “tent” function (also known as a *Bartlett* filter). Figure 3.13b shows a 3×3 version of this filter, which is called the *bilinear* kernel, since it is the outer product of two linear (first order) splines (§3.4.1).

Convolving the linear tent function with itself yields the cubic approximating spline, which is called the “Gaussian” kernel (Figure 3.13c) in Burt and Adelson’s (1983a) *Laplacian pyramid* representation (§3.4). Note that approximate Gaussian kernels can also be obtained by iterated convolution with box filters (Wells 1986). In applications where the filters really need to be rotationally symmetric, carefully tuned versions of sampled Gaussians should be used (Freeman and Adelson 1991) (Exercise 3.10).

The kernels I just discussed are all examples of blurring (smoothing) or *low-pass* kernels (since they pass through the lower frequencies while attenuating higher frequencies). How good are they at doing this? In §3.3, I use frequency-space Fourier analysis to examine the exact frequency response of these filters. I also introduce the *sinc* ($(\sin x)/x$) filter, which performs *ideal* low-pass filtering.

In practice, smoothing kernels are often used to reduce high-frequency noise. I will have much more to say about using variants on smoothing to remove noise later in this book (§3.2.2, §3.3, and §3.6).

Surprisingly, smoothing kernels can also be used to *sharpen* images using a process called *unsharp masking*. Since blurring the image reduces high frequencies, adding some of the difference between the original and the blurred image makes it sharper,

$$g_{\text{sharp}} = f + \gamma(f - h_{\text{blur}} * f). \quad (3.22)$$

In fact, before the advent of digital photography, this was the standard way to sharpen images in the darkroom: create a blurred (“positive”) negative from the original negative by mis-focusing, then overlay the two negatives before printing the final image, which corresponds to

$$g_{\text{unsharp}} = f(1 - \gamma h_{\text{blur}} * f). \quad (3.23)$$

(This is no longer a linear filter, but still works well.)

Linear filtering can also be used as a pre-processing stage to edge extraction §4.2 and interest point detection §4.1 algorithms. Figure 3.13c shows a simple 3×3 edge extractor called the Sobel

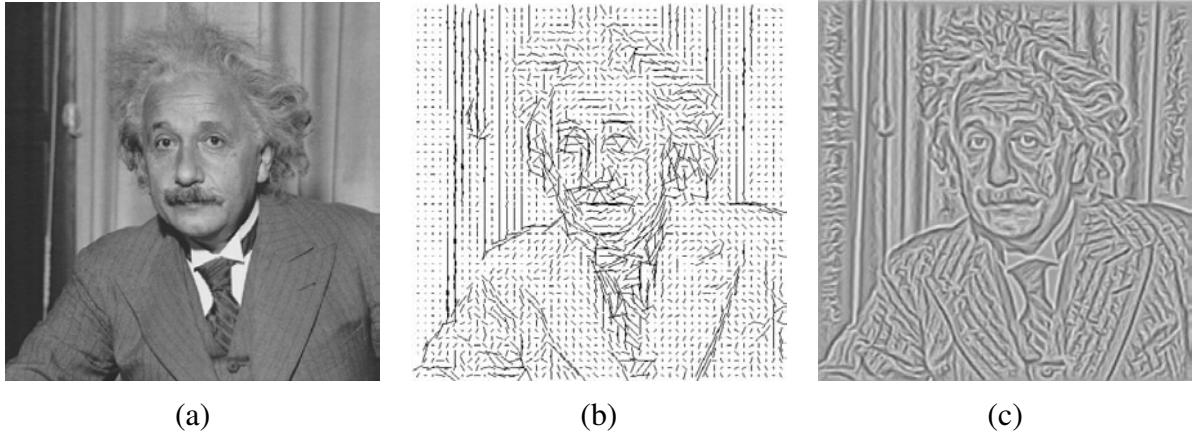


Figure 3.14: *Second order steerable filter (Freeman 1992)*: (a) original image of Einstein; (b) orientation map computed from the second order oriented energy; (c) original image with oriented structures enhanced.

operator, which is a separable combination of a horizontal *central difference* (so called because the horizontal derivative is centered on the pixel) and a vertical box filter (to smooth the results). As you can see in the image below the kernel, this filter effectively emphasizes horizontal edges.

The 5-point “Laplacian” operator next to it looks for simultaneous horizontal and vertical derivatives (or rather, derivatives at different orientations). It is a simple example of what is sometimes called a “corner detector” (§4.1), since, it responds well at rectangle and triangle corners, while largely ignoring oriented edges.

Band-pass and steerable filters

The 5-point Laplacian and Sobel operators are simple examples of band-pass and oriented filters. More sophisticated kernels can be created by first smoothing the image with a (unit area) Gaussian filter,

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad (3.24)$$

and then taking the first or second derivatives (Marr 1982, Witkin 1983, Freeman and Adelson 1991). Such filters are known collectively as *band-pass filters*, since they filter away the low frequencies (see Tables 3.2 and 3.3).

The (undirected) second derivative of a two dimensional image,

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}, \quad (3.25)$$

is known as the *Laplacian* operator. Blurring an image with a Gaussian and then taking its Lapla-

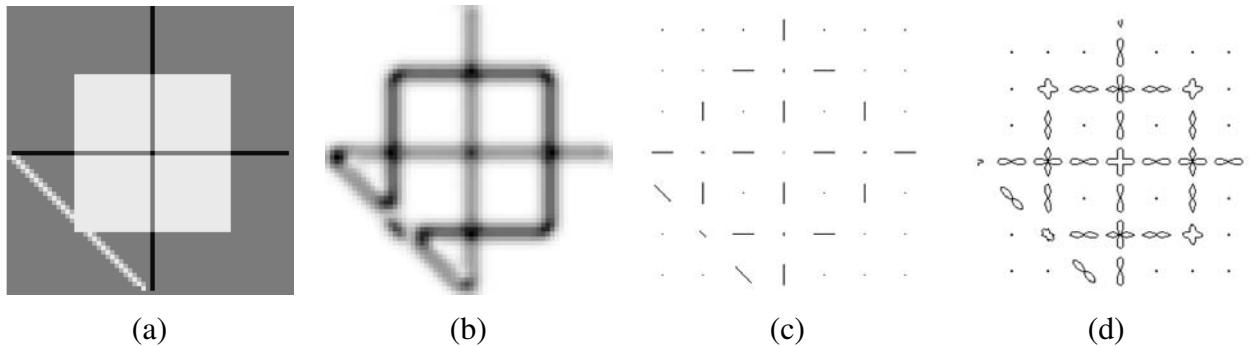


Figure 3.15: *Fourth order steerable filter (Freeman and Adelson 1991)*: (a) test image containing bars (lines) and step edges at different orientations; (b) average oriented energy; (c) dominant orientation; (d) oriented energy as a function of angle (polar plot).

cian is equivalent to convolving directly with the *Laplacian of Gaussian* (LoG) filter,

$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{1}{\sigma^2} \right) G(x, y; \sigma), \quad (3.26)$$

which has certain nice *scale space properties* (Witkin 1983, Witkin *et al.* 1986). The 5-point Laplacian is just a compact approximation to this more sophisticated filter.

Likewise, the Sobel operator is a simple approximation to a *directional* or *oriented* filter, which can be obtained by smoothing with a Gaussian (or some other filter) and then taking a *directional derivative* $\nabla_{\hat{\mathbf{u}}} = \frac{\partial}{\partial \hat{\mathbf{u}}}$, which is obtained by taking the dot product between the gradient field ∇ and a unit direction $\hat{\mathbf{u}} = (\cos \theta, \sin \theta)$,

$$\hat{\mathbf{u}} \cdot \nabla(G * f) = \nabla_{\hat{\mathbf{u}}}(G * f) = (\nabla_{\hat{\mathbf{u}}} G) * f. \quad (3.27)$$

The smoothed directional derivative filter,

$$G_{\hat{\mathbf{u}}} = uG_x + vG_y = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y}, \quad (3.28)$$

where $\hat{\mathbf{u}} = (u, v)$, is an example of a *steerable* filter, since the value of an image convolved with $G_{\hat{\mathbf{u}}}$ can be computed by first convolving with the pair of filters (G_x, G_y) and then *steering* the filter (potentially locally) by multiplying this gradient field with a unit vector $\hat{\mathbf{u}}$ (Freeman and Adelson 1991). The advantage of this approach is that a whole *family* of filters can be evaluated with very little cost.

How about steering a directional second derivative filter $\nabla_{\hat{\mathbf{u}}} \cdot \nabla_{\hat{\mathbf{u}}} G_{\hat{\mathbf{u}}}$, which is the result of taking a (smoothed) directional derivative, and then taking the directional derivative again? (For example, G_{xx} is the second directional derivative in the x direction.)

At first glance, it would appear that the steering trick will not work, since for every direction \hat{u} , we need to compute a different first directional derivative. Somewhat surprisingly, [Freeman and Adelson \(1991\)](#) showed that for directional Gaussian derivatives, it is possible to steer *any* order of derivative with a relatively small number of basis functions. For example, only 3 basis functions are required for the second order directional derivative,

$$G_{\hat{u}\hat{u}} = u^2 G_{xx} + 2uv G_{xy} + v^2 G_{yy}. \quad (3.29)$$

Furthermore, each of the basis filters, while itself not necessarily separable, can be computed using a linear combination of a small number of separable filters ([Freeman and Adelson 1991](#)).

This remarkable result makes it possible to construct directional derivative filters of increasingly greater *directional selectivity*, i.e., filters that only respond to edges that have strong local consistency in orientation (Figure 3.14). Furthermore, higher order steerable filters can respond to potentially more than a single edge orientation at a given location, and they can respond to both *bar* edges (thin lines), as well as the classic step edges (Figure 3.15). In order to do this, however, full *Hilbert transform pairs* need to be used for second order and higher filters, as described in ([Freeman and Adelson 1991](#)).

Steerable filters are often used to construct both feature descriptors §4.1.3 and edge detectors §4.2. While the filters developed by [Freeman and Adelson \(1991\)](#) are best suited for detecting linear (edge-like) structures, recent work by [Koethe \(2003\)](#) shows how a combined 2×2 boundary tensor can be used to encode both edge and junction (“corner”) features. Exercise 3.12 has you implement such steerable filters and apply them to finding both edge and corner features.

Summed area tables [optional]

If an image is going to be repeatedly convolved with different box filters (and especially filters of different size at different locations), you can precompute the *summed area table* ([Crow 1984](#)), which is just the running sum of all the pixel values from the origin,

$$s(i, j) = \sum_{k=0}^i \sum_{l=0}^j f(k, l). \quad (3.30)$$

This can be efficiently computed using a recursive (raster-scan) algorithm,

$$s(i, j) = s(i - 1, j) + s(i, j - 1) - s(i - 1, j - 1) + f(i, j). \quad (3.31)$$

To find the summed area inside a rectangle $[i_0, i_1] \times [j_0, j_1]$, we simply combine four samples from the summed area table,

$$S(i_0 \dots i_1, j_0 \dots j_1) = \sum_{i=i_0}^{i_1} \sum_{j=j_0}^{j_1} s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1). \quad (3.32)$$

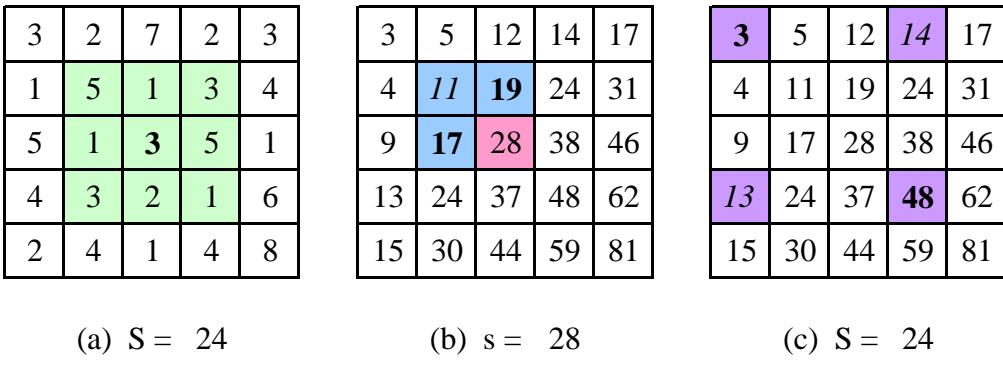


Figure 3.16: *Summed area tables*: (a) original image; (b) summed area table; (c) computation of area sum. Each value in the summed area table (red) is computed recursively from its three adjacent (blue) neighbors. Area sums (green) are computed by combining the four values at the rectangle corners (purple). (Positive value are shown in **bold** and negative values in italics.)

A potential disadvantage of summed area tables is that they require $\log M + \log N$ extra bits in the accumulation image compared to the original image, where M and N are the image width and height. Extensions of summed area tables can also be used to approximate other convolution kernels (see (Wolberg 1990, §6.5.2) for a review).

In computer vision, summed area tables have been used in face detection (Viola and Jones 2004) to compute simple multi-scale low-level features. Such features, which consist of adjacent rectangles of positive and negative values, are also known as *boxlets* (Simard *et al.* 1998). In principle, summed area tables could also be used to compute the sums in the sum-of-squared difference (SSD) stereo and motion algorithms §10.4; in practice, separable moving average filters are usually preferred (Kanade *et al.* 1996).

Recursive filtering [optional]

The incremental formula (3.31) for the summed area is an example of a *recursive filter*, i.e., one whose values depends on previous filter outputs. In the signal processing literature, such filters are known as *infinite impulse response* (IIR), since the output of the filter to an impulse (single non-zero value) goes on forever. (For the summed area table, an impulse generates an infinite rectangle of 1s below and to the right of the impulse.) The filters we have previously studied in this chapter, which involve the image with a finite extent kernel, are known as *finite impulse response* (FIR).

Two-dimensional IIR filters and recursive formulas are sometimes used to compute quantities that involve large area interactions, such as two-dimensional distance functions (§3.2.4) and connected components (§3.2.5).

More commonly, however, IIR filters are used inside one-dimensional separable filtering stages

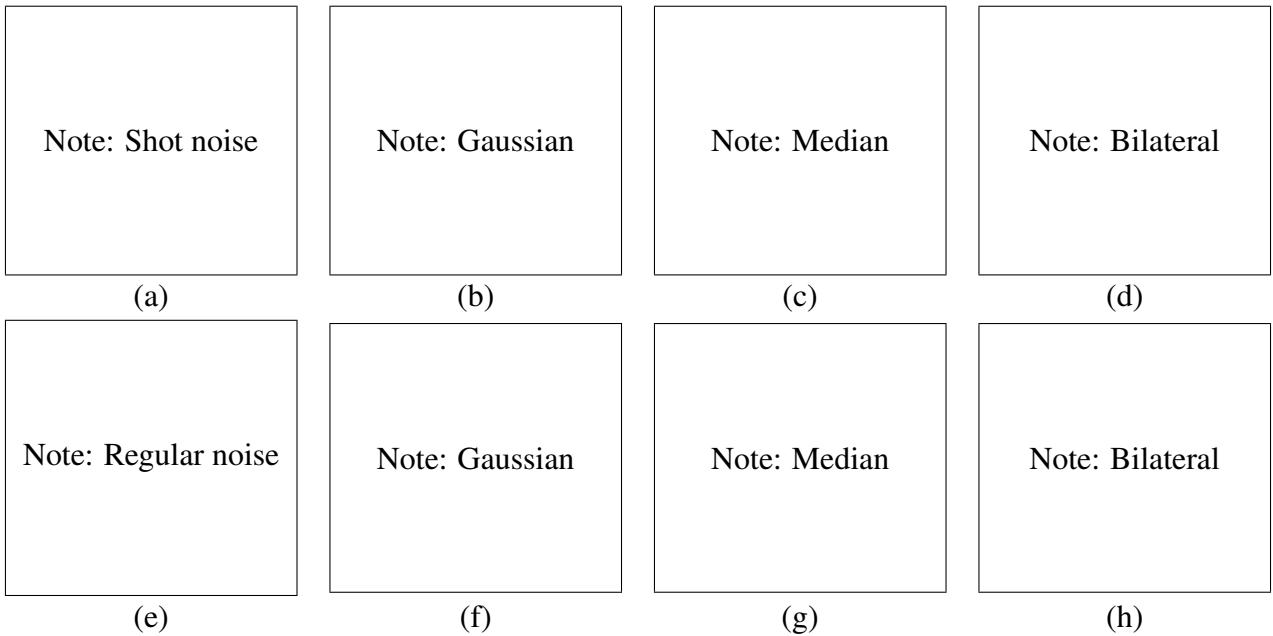


Figure 3.17: *Median and bilateral filtering*: (a) image with shot noise; (b) Gaussian filtered; (c) median filtered; (d) bilaterally filtered; (e) image with regular noise; (f) Gaussian filtered; (g) median filtered; (h) bilaterally filtered.

to compute large-extent smoothing kernels, such as efficient approximations to Gaussians and edge filters (Deriche 1990, Nielsen *et al.* 1997). Pyramid-based algorithms (§3.4) can also be used to perform such large-area smoothing computations.

3.2.2 Non-linear filtering

The filters we have looked at so far have all been *linear*, i.e., their response to a sum of two signals is the same as the sum of the individual responses. This is equivalent to saying that each output pixel is a weighted summation of some number of input pixels (3.19). Linear filters are easier to compose and are amenable to frequency response analysis §3.3.

In many cases, however, better performance can be obtained by using a *non-linear* combination of neighboring pixels. Consider for example the image in Figure 3.17a, where the noise, rather than being Gaussian, is *shot noise*, i.e., it occasionally has very large values. In this case, regular blurring with a Gaussian filter fails to remove the noisy pixels, and instead turns them into softer (but still visible) spots.

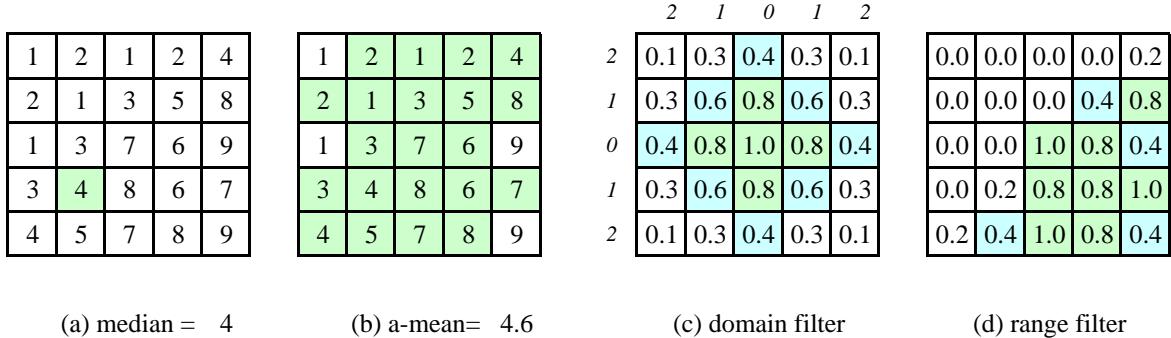


Figure 3.18: *Median and bilateral filtering:* (a) median pixel shown in green; (b) selected α -trimmed mean pixels; (c) domain filter (numbers along edge are pixel distances); (d) range filter.

Median filtering

A better filter to use in this case is the *median* filter, which selects the median value from each pixels' neighborhood (Figure 3.18a). Median values can be computed in expected linear time using the randomized select algorithm described in (Cormen 2001), and incremental variants have also been developed [Note: Add citations from (Tomasi and Manduchi 1998) and/or cite (Bovik 2000, §3.2).] Since the shot noise value usually lies well outside of the true values in the neighborhood, the median filter is able to filter away such bad pixels (Figure 3.17c).

One downside of the median filter, in addition to its moderate computational cost, is that since it selects only one input pixel value to replace each output pixel, it is not as *efficient* at averaging away regular Gaussian noise (see the discussion of efficiency in Appendix B.3 on robust statistic as well as textbooks on this subject (Huber 1981, Hampel *et al.* 1986, Stewart 1999)). A better choice may be the α -trimmed mean [Note: (Crane 1997, p. 109), e.g., Lee CG&A May 1990], which averages together all of the pixels except for the α fraction that are the smallest and the largest (Figure 3.18b).

Another possibility is to compute a *weighted median*, in which each pixel is used a number of times depending on its distance to the center. This turns out to be equivalent to minimizing the weighted objective function

$$\sum_{k,l} w(k,l) |f(i+k, j+l) - g(i, j)|^p, \quad (3.33)$$

where $g(i, j)$ is the desired output value, and $p = 1$ for the weighted median. (The value $p = 2$ is the usual *weighted mean*, which is equivalent to correlation (3.12) after normalizing by the sum of the weights.) The weighted mean therefore has deep connections to other methods in robust statistics (Appendix B.3) such as influence functions (Huber 1981, Hampel *et al.* 1986). [Note: There's a nice discussion of weighted median filtering in (Bovik 2000, §3.2) as well as (Haralick

and Shapiro 1992, §7.2.6), which also discusses the α -trimmed mean. If I need more citations to original work (dating back a century!), it's all in there.]

Non-linear smoothing has another, perhaps even more important property (especially since shot noise is rare in today's cameras). Such filtering is more *edge preserving*, i.e., it has less tendency to soften edges while filtering away high-frequency noise.

Consider the noisy image in Figure 3.17e. In order to remove most of the noise, the Gaussian filter is forced to smooth away high-frequency detail, which is most noticeable near strong edges. Median filtering does better, but as mentioned before, does not do as good a job at smoothing away from discontinuities. (See (Tomasi and Manduchi 1998) for some additional references to edge preserving smoothing techniques.)

While we could try to use the α -trimmed mean or weighted median, these techniques still all have a tendency to round sharp corners, since the majority of pixels in the smoothing area come from the background distribution.

Bilateral filtering

What if we combined the idea of a weighted filter kernel with a better version of outlier rejection? What if instead of rejecting a fixed percentage α , we simply rejected (in a soft way), pixels whose *values* differed too much from the central pixel value? This is the essential idea in *bilateral filtering*, which was first popularized in the computer vision community by Tomasi and Manduchi (1998). (See (Chen *et al.* 2007) for citations of similar earlier (Aurich and Weule 1995, Smith and Brady 1997) as well as the wealth of subsequent applications in computer vision and computational photography.)

In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values

$$g(i, j) = \frac{\sum_{k,l} f(k, l)w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}. \quad (3.34)$$

The weighting coefficient $w(i, j, k, l)$ depends on the product of a *domain kernel* (Figure 3.18c),

$$d(i, j, k, l) = \exp\left(-\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2}\right), \quad (3.35)$$

and a data-dependent *range kernel* (Figure 3.18d),

$$r(i, j, k, l) = \exp\left(-\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right). \quad (3.36)$$

[Note: This may be the only place where I use \exp in this chapter. Use the e^{\dots} notation instead? I just hate how small everything then looks .] When multiplied together, these yield the data-

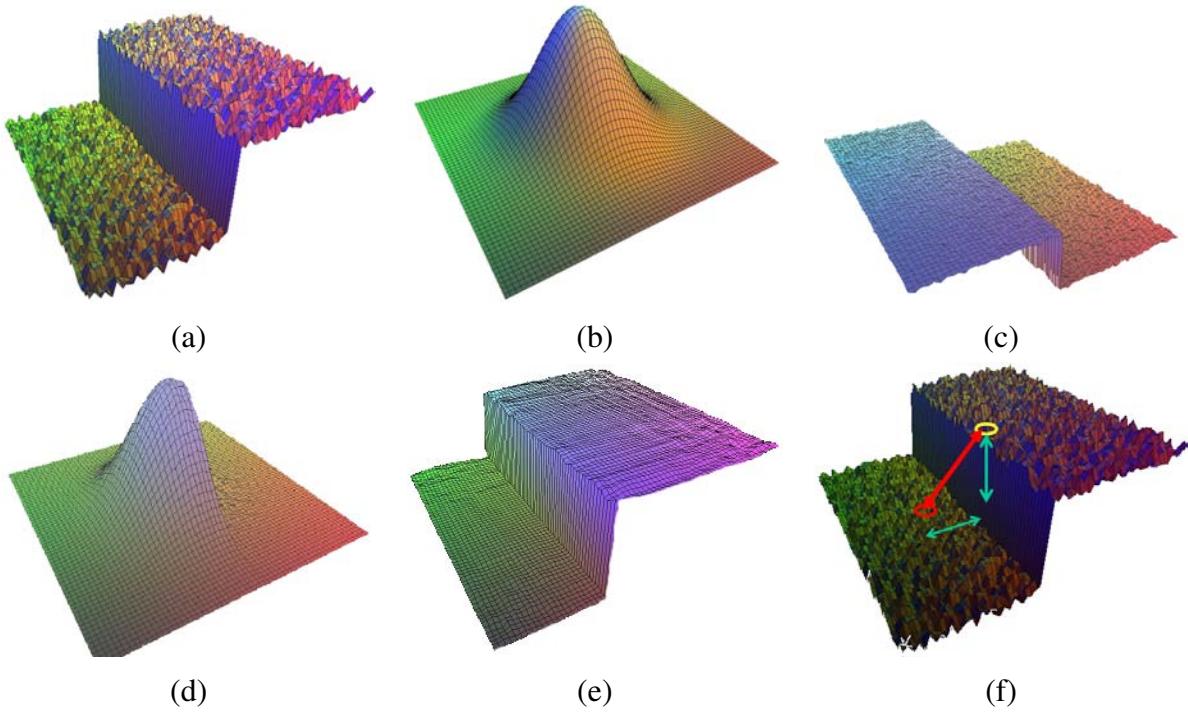


Figure 3.19: *Bilateral filtering* (Durand and Dorsey 2002): (a) noisy step edge input; (b) domain filter (Gaussian); (c) range filter (similarity to center pixel value); (d) bilateral filter; (e) filtered step edge output; (f) 3D distance between pixels.

dependent *bilateral weight function*

$$w(i, j, k, l) = \exp \left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right). \quad (3.37)$$

Figure 3.19 shows an example of a noisy step edge being bilaterally filtered. Note how the domain kernel is the usual Gaussian, the range kernel measures appearance (intensity) similarity to the center pixel, and the bilateral filter kernel is a product of these two.

Notice that the range filter above used the *vector distance* between the center and neighboring pixel. This is important in color images, since an edge in any *one* of the color bands signals a change in material, and hence the need to downweight a pixel's influence.⁴

Since bilateral filtering is quite slow compared to regular separable filtering, a number of acceleration techniques have been developed (Durand and Dorsey 2002, Paris and Durand 2006, Chen *et al.* 2007). Unfortunately, these techniques tend to use more memory than regular filtering, and are hence not directly applicable to full color image filtering.

⁴ Tomasi and Manduchi (1998) show that using the vector distance (as opposed to separately filtering each color band separately) reduces color fringing effects. They also recommend taking the color difference in the more perceptually uniform CIE-Lab color space §2.3.2.

Iterated adaptive smoothing and anisotropic diffusion

Bilateral (and other) filters can also be applied in an iterative fashion, especially if a more “cartoon” like appearance is desired (Tomasi and Manduchi 1998). When iterated filtering is applied, a much smaller neighborhood can often be used.

Consider, for example, using only the four nearest neighbors, i.e., restricting $|k-i| + |l-j| \leq 1$ in (3.34). Observe that

$$d(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right) = \begin{cases} 1, & |k-i| + |l-j| = 0, \\ \lambda = e^{-1/2\sigma_d^2}, & |k-i| + |l-j| = 1. \end{cases} \quad (3.38)$$

We can thus re-write (3.34) as

$$\begin{aligned} f^{(t+1)}(i, j) &= \frac{f^{(t)}(i, j) + \eta \sum_{k,l} f^{(t)}(k, l) r(i, j, k, l)}{1 + \eta \sum_{k,l} r(i, j, k, l)} \\ &= f^{(t)}(i, j) + \frac{\eta}{1 + \eta R} \sum_{k,l} r(i, j, k, l) [f^{(t)}(k, l) - f^{(t)}(i, j)], \end{aligned} \quad (3.39)$$

where $R = \sum_{k,l} r(i, j, k, l)$, (k, l) are the \mathcal{N}_4 neighbors of (i, j) , and we have made the iterative nature of the filtering explicit.

As Barash (2002) notes, (3.39), is the same as the discrete *anisotropic diffusion* equation first proposed by Perona and Malik (1990b).⁵ Since its original introduction, anisotropic diffusion has been extended and applied to a wide range of problems (Nielsen *et al.* 1997, Black *et al.* 1998, Weickert *et al.* 1998, Weickert 1998) and also shown to be closely related to other *adaptive smoothing* techniques (Saint-Marc *et al.* 1991, Barash 2002).

In its general form, the range kernel $r(i, j, k, l) = r(\|f(i, j) - f(k, l)\|)$, which is usually called the *gain* or *edge-stopping* function or diffusion coefficient, can be any monotonically increasing function with $r'(x) \rightarrow 0$ as $x \rightarrow \infty$. Black *et al.* (1998) show how anisotropic diffusion is equivalent to minimizing a robust penalty function on the image gradients, e.g., (3.104) and (3.112), which we will discuss in the sections on robust regularization §3.6.1 and Markov random field (MRF) priors §3.6.2.

Note that without a bias term towards the original image, anisotropic diffusion and iterative adaptive smoothing techniques will converge to a constant image. Unless a small number of iterations is going to be used (e.g., for speed), it is usually preferable to formulate the smoothing problem as a joint minimization of a smoothness term and a data fidelity term, as discussed in §3.6.1 and §3.6.2, which introduces such a bias in a principled manner.

[Note: Read the following (*Scharr-iccv2003.pdf*) and see if it fits in: *Image Statistics and Anisotropic Diffusion* (*Scharr et al. 2003*): relationship between MRF / Laplacian statistics and anisotropic diffusion.]

⁵ The $1/(1 + \eta R)$ factor is not present in anisotropic diffusion, but becomes negligible as $\eta \rightarrow 0$.

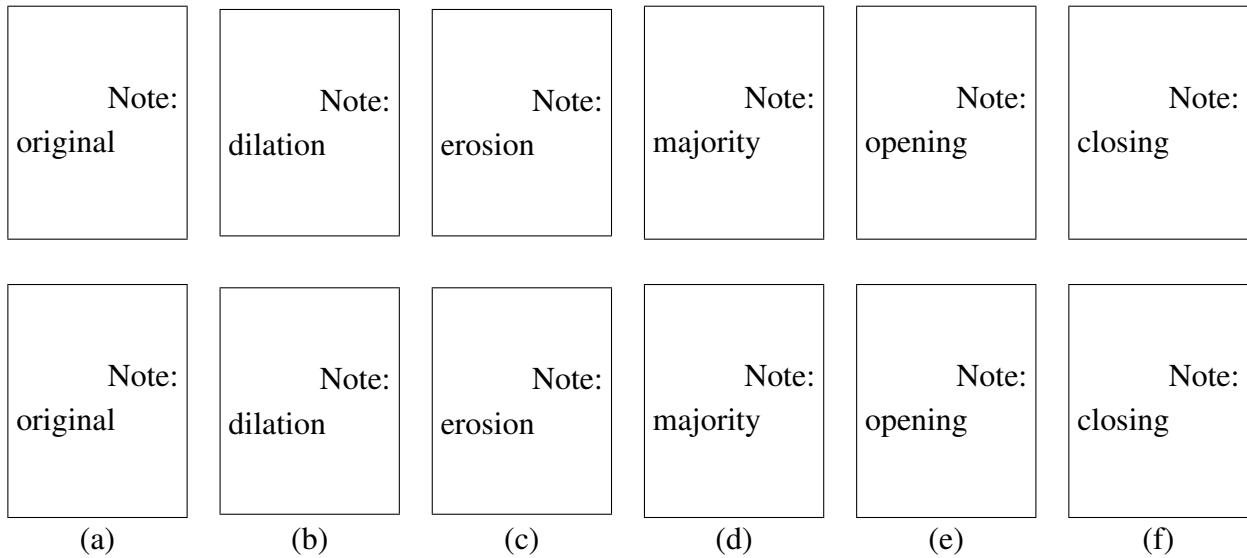


Figure 3.20: *Binary image morphology*: (a) *original image with structuring element overlaid*; (b) *dilation*; (c) *erosion*; (d) *majority*; (e) *opening*; (f) *closing*.

[Note: Top row: generate a set of close-up images in Excel, or import the close-up outputs from a text-separated crop of the full image (bottom row) computation.]

3.2.3 Morphology

While non-linear filters are often used to enhance grayscale and color images, they are also used extensively to process binary images. Such images often occur after a *thresholding* operation,

$$\theta(f, c) = \begin{cases} 1 & \text{if } f > c, \\ 0 & \text{else,} \end{cases} \quad (3.40)$$

e.g., converting a scanned grayscale document into a binary image for further processing such as *optical character recognition*.

The most common binary image operations are called *morphological operations*, since they change the *shape* of the underlying binary objects (Ritter and Wilson 2000, §7). To perform such an operation, we first convolve the binary image with a binary *structuring element* and then select a binary output value depending on the thresholded result of the convolution. (This is not the usual way in which these operations are described, but I find it a nice simple way to unify the processes.) The structuring element can be any shape, from a simple 3×3 box filter, to more complicated disc structures. It can even correspond to a particular shape that is being sought for in the image.

Figure 3.20 shows a close-up of the convolution of a binary image f with a 3×3 structuring element s , and the resulting images for the operations described below. Let

$$c = f \otimes s \quad (3.41)$$

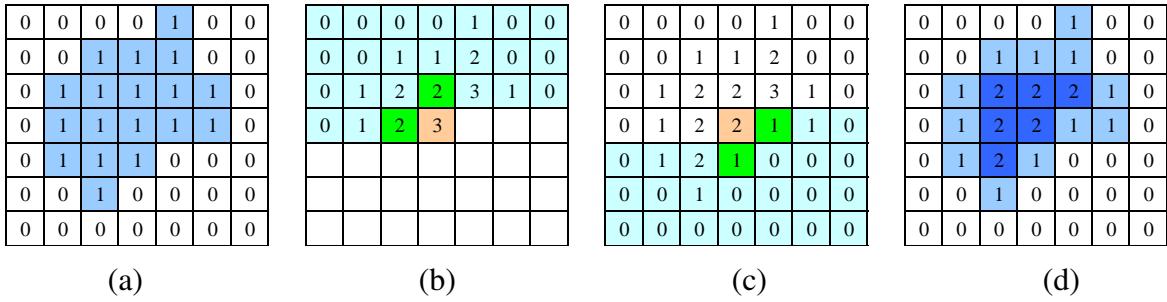


Figure 3.21: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange one; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

be the integer-valued *count* of the number of 1s inside each structuring element as it is scanned over the image, and S be the size of the structuring element (number of pixels). The standard operations used in binary morphology include:

- **dilation:** $\text{dilate}(f, s) = \theta(c, 0);$
 - **erosion:** $\text{erode}(f, s) = \theta(c, S);$
 - **majority:** $\text{maj}(f, s) = \theta(c, S/2);$
 - **opening:** $\text{open}(f, s) = \text{dilate}(\text{erode}(f, s), s);$
 - **closing:** $\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s).$

As we can see from Figure 3.20, dilation grows (thickens) objects consisting of 1s, while erosion shrinks (thins) them. The opening and closing operations tend to leave large regions and smooth boundaries unaffected, while removing small objects or holes and smoothing boundaries.

While we will not use mathematical morphology much in the rest of this book, it's a handy tool to have around whenever you need to clean up some thresholded images. You can find additional details on morphology in other textbooks on computer vision ([Bovik 2000](#), §2.2), ([Haralick and Shapiro 1992](#), §5.2), as well as articles and books specifically on this topic ([Serra 1982](#), [Soille 2006](#)). [Note: Add some papers by Vincent? Check CV bibliography.]

3.2.4 Distance transforms

The distance transform is useful in quickly precomputing the distance to a curve or set of points using a two-pass raster algorithm (Danielsson 1980, Borgefors 1986, Paglieroni 1991, Huttenlocher *et al.* 1993). It has many applications, including level sets §4.4.3, fast *chamfer matching* (binary

image alignment) (Huttenlocher *et al.* 1993), feathering in image stitching and blending §8.3.2, and nearest point alignment §11.4.1.

The distance transform $D(i, j)$ of a binary image $b(i, j)$ is defined as follows. Let $d(k, l)$ be some *distance metric* between pixel offsets. Two commonly used metrics include the *city block* or *Manhattan* distance

$$d_1(k, l) = |k| + |l| \quad (3.42)$$

and the *Euclidean* distance

$$d_2(k, l) = \sqrt{k^2 + l^2}. \quad (3.43)$$

The distance transform is then defined as

$$D(i, j) = \min_{k, l: b(k, l)=0} d(i - k, j - l), \quad (3.44)$$

i.e., it is the distance to the *nearest* background pixel whose value is 0.

The D_1 city block distance transform can be efficiently computed using a forward and backward pass of a simple raster-scan algorithm, as shown in Figure 3.21. During the forward pass, each non-zero pixel in b is replaced by the minimum of $1 +$ the distance of its north or west neighbor. During the backward pass, the same occurs, except that the minimum is both over the current value D and $1 +$ the distance of the south and east neighbors (Figure 3.21).

Efficiently computing the Euclidean distance transform is more complicated. Here, just keeping the minimum scalar distance to the boundary during the two passes is not sufficient. Instead, a *vectorial* distance consisting of both the x and y coordinates of the distance to the boundary must be kept and compared together using the squared distance (hypotenuse) rule. As well, larger search regions need to be used to obtain reasonable results. Rather than explaining the algorithm in more detail (Danielsson 1980, Borgefors 1986), I leave it as an Exercise 3.13 for the motivated reader.

[*Note: Finish up the rest of this subsection...*]

[*Note: Have a look at Pedro's dt Web page, <http://people.cs.uchicago.edu/~pff/dt/>.*]

Figure ... shows a variety of distance transforms applied to a sample binary image. [*Note: See if (Rosenfeld and Kak 1976) has these, or if there are pictures in (Danielsson 1980, Borgefors 1986) J.* Grassfire transform, connection to repeated erosion.

Keeping the Vectorial output around is useful for nearest border matching / ICP [Lavallée] (move up?)

Signed distance useful to compute inside/outside distance (Lavallée and Szeliski 1995, Szeliski and Lavallée 1996) [*Note: Add adaptive distance fields, (Friskin et al. 2000), as well as merging range data (Curless and Levoy 1996).*] Also, connection to level-set theory §4.4.3 (already said this in section introduction).

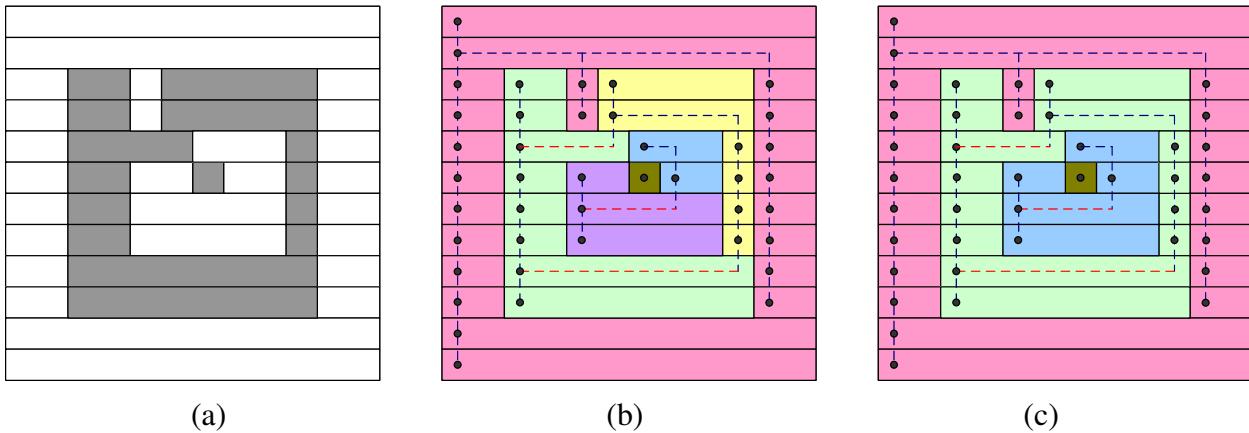


Figure 3.22: *Connected component computation:* (a) original grayscale image; (b) horizontal runs (nodes) connected by vertical (graph) edges (dashed blue); runs are pseudocolored with unique colors inherited from parent nodes; (c) re-coloring after traversing the forest of trees.

[Note: This diagram is misleading, since it suggests that each graph must be a tree. In fact, if you join the spiral arms together, you get a DAG. May want to revise this diagram. In fact, I've started making the changes, but have not finished yet or updated the PDF.]

3.2.5 Connected components

[Note: I wrote this section before looking at (Haralick and Shapiro 1992, §2.3) and now realize that there are simpler variants available. Should I bother modifying the figure to only show connections between items in the equivalence class table (which is potentially quadratic in the number of components), and modify my description (no need for pointers)? I think it's o.k. to leave as is, but if anyone finds a bug in my high-level description, I'll need to fix this. Or, when I get more time, just re-read (Haralick and Shapiro 1992, §2.3) and re-do this section.]

Another useful semi-global image transform is finding *connected components*, which are defined as regions of adjacent pixels that have the same input value (or label). (In the remainder of this section, consider pixels to be *adjacent* if they are immediate \mathcal{N}_4 neighbors and they have the same input value.) Connected components can be used in a variety of applications, such as finding individual letters in a scanned document, to finding objects (say cells) in a thresholded image and computing their area statistics (see below).

Consider the grayscale image in Figure 3.22a. There are four connected components in this figure: the outermost set of white pixels, the large ring of gray pixels, the white enclosed region, and the single gray pixel. These are shown pseudocolored in Figure 3.22c as pink, green, blue, and brown.

To compute the connected components of an image, we first (conceptually) split the image into horizontal *runs* of adjacent pixels, and then color the runs with unique labels, re-using the labels

of vertically adjacent runs whenever possible.

Figure 3.22b shows in more detail how the process works. In this figure, a dot is placed at the start of each horizontal run to indicate a node in the graph of connected components. (This graph, as we will see, will be a forest of trees.) To begin the process, we assign a unique label (pink) to the first (upper left) pixel. (I use colors in the description below, but in practice, a counter is used to assign unique label values.)

We run along all the pixels in a run, copying the color of that run, or (better yet), storing back pointers to the head of the run, which contains the unique label (this allows the information to be stored “in-place” in a working image). When we hit a new run (change of input labels), we either take the value of the (same input) adjacent run in the previous row, or start a new unique id. Figure 3.22b shows the former case using a dashed blue line to indicate when one run (node circle) takes the value of a previous run (in practice, it stores a back pointer to the previous run).

Sometimes, as is the case of the green and yellow or purple and blue runs that are vertically adjacent, a non-head pixel will be adjacent to a differently colored run. In this case, we would like the component colored with the higher label to take on the color of the lower label. We can store such re-coloring commands in a separate table, indexed by the component colors. In Figure 3.22b, we show such relationships schematically as red dashed lines.

During the second phase of the algorithm, re-colorings are recursively updated by walking through the re-coloring table from smallest to largest label. Since a re-coloring always points from a higher label value to a lower one, this effectively merges all trees (connected components) that are adjacent into a single component. A final phase can then go back and color the output pixels with their correct (re-colored) values, as shown in Figure 3.22c.

While this description is a little sketchy, it should be enough to enable a motivated student to implement this algorithm (Exercise 3.14). Haralick and Shapiro (1992), §2.3 contain a much longer description of other connected component algorithms, including ones which avoid the creation of a potentially large re-coloring (equivalence) table. Well debugged connected component algorithms are also available in most image processing libraries.

Area statistics

Once a binary or multi-valued image has been segmented into its connected components, it is often useful to compute the area statistics for each individual region \mathcal{R} . Such statistics include:

- the area (number of pixels);
- the perimeter (number of boundary pixels);
- the centroid (average x and y values);

- the second moments,

$$\mathbf{M} = \sum_{(x,y) \in \mathcal{R}} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix}, \quad (3.45)$$

from which the major and minor axis orientation and lengths can be computed using eigenvalue analysis.

These statistics can then be used for further processing, e.g., for sorting the regions in decreasing area (to consider the largest regions first) or for doing preliminary matching between regions in different images.

3.3 Fourier transforms

In the previous section §3.2.1, I mentioned that Fourier analysis could be used to analyze the frequency characteristics of various filters. In this section, I explain both how Fourier analysis lets us determine these characteristics (or equivalently, the frequency *content* of an image), and how using the Fast Fourier Transform (FFT) lets us perform large-kernel convolutions in time that is independent of the kernel's size. (For a more comprehensive introduction to Fourier transforms, see (Bracewell 1986, Glassner 1995, Oppenheim and Schafer 1996, Oppenheim *et al.* 1999).)

How can we analyze what a given filter does to high, medium, and low frequencies? The answer is to simply pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated. Let

$$s(x) = \sin(2\pi f x + \phi_i) = \sin(\omega x + \phi_i) \quad (3.46)$$

be the input sinusoid whose *frequency* is f , *angular frequency* is $\omega = 2\pi f$, and *phase* is ϕ_i . Note that in this section, I use the variables x and y to denote the spatial coordinates of an image, rather than i and j as in the previous sections. This is both because the letters i and j are used for the *imaginary* number (the usage depends on whether you are reading the complex variables or electrical engineering literature), and because it is clearer how to distinguish the horizontal (x) and vertical (y) components in the frequency space. In this section, I use the letter j for the imaginary number, since that is the form more commonly found in the signal processing literature (Bracewell 1986, Oppenheim and Schafer 1996, Oppenheim *et al.* 1999).

[Note: I initially wrote this section using f for frequency, but I've mostly switched to ω for angular frequency. Some authors also use u and v for horizontal and vertical frequency, but I prefer to reserve those for optical flow.]

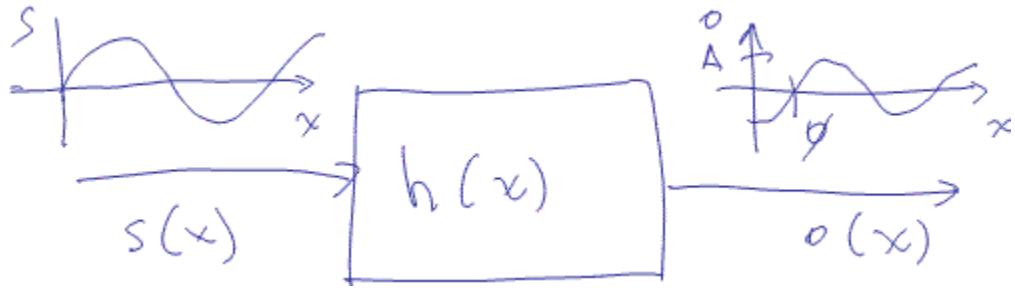


Figure 3.23: *The Fourier Transform as the response of a filter $h(x)$ to an input sinusoid $s(x) = e^{j\omega x}$ yielding an output sinusoid $o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$.*

If we convolve the sinusoidal signal $s(x)$ with a filter whose impulse response is $h(x)$, we get another sinusoid of the same frequency but different magnitude A and phase ϕ_o ,

$$o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o) \quad (3.47)$$

(Figure 3.23). To see that this is the case, remember that a convolution can be expressed as a weighted summation of shifted input signals (3.14), and that the summation of a bunch of shifted sinusoids of the same frequency is just a single sinusoid at that frequency.⁶ The new magnitude A is called the *gain* or *magnitude* of the filter, while the phase difference $\Delta\phi = \phi_o - \phi_i$ is called the *shift* or *phase*.

In fact, a more compact notation is to use the complex-valued sinusoid

$$s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x. \quad (3.48)$$

In that case, we can simply write,

$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}. \quad (3.49)$$

The *Fourier transform* is simply a tabulation of the magnitude and phase response to each frequency,

$$H(\omega) = \mathcal{F}\{h(x)\} = Ae^{j\phi}, \quad (3.50)$$

i.e., it is the response to a complex sinusoid of frequency ω passed through the filter $h(x)$. The Fourier transform pair is also often written as

$$h(x) \xleftrightarrow{\mathcal{F}} H(\omega). \quad (3.51)$$

⁶ If h is a general (non-linear) transform, additional *harmonic* frequencies will be introduced. This was traditionally the bane of audiophiles, who insisted on equipment with no *harmonic distortion*. Now that digital audio has introduced pure distortion-free sound, some audiophiles are buying retro tube amplifiers or digital signal processors that simulate such distortions because of their “warmer sound”.

Unfortunately, (3.50) does not give an actual *formula* for computing the Fourier transform. (Instead, it gives a *recipe*, i.e., convolve the filter with a sinusoid, observe the magnitude and phase shift, repeat.) Fortunately, closed form equations for the Fourier transform exist both in the continuous domain,

$$H(\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x} dx, \quad (3.52)$$

and in the discrete domain,

$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi k x}{N}}, \quad (3.53)$$

where N is the length of the signal or region of analysis. These formulas apply both to filters, such as $h(x)$ and to signals or images, such as $s(x)$ or $g(x)$.

The discrete form of the Fourier transform (3.53) is known as the *Discrete Fourier Transform* (DFT). Note that while (3.53) can be evaluated for any value of k , it only makes sense for values in the range $k \in [-\frac{N}{2}, \frac{N}{2}]$. This is because larger values of k alias with lower frequencies, and hence provide no additional information, as explained in the previous discussion on aliasing §2.3.1.

At face value, the DFT takes $O(N^2)$ operations (multiply-adds) to evaluate. Fortunately, there exists a faster algorithm called the *Fast Fourier Transform* (FFT), which only requires $O(N \log_2 N)$ operations (Bracewell 1986, Oppenheim *et al.* 1999). I will not explain the details of the algorithm here, except to say that it involves a series of $\log_2 N$ stages, where each stage performs small 2×2 transforms (matrix multiplies with known coefficients) followed by some semi-global permutations. (You will often see the term *butterfly* applied to these stages because of the pictorial shape of the signal processing graphs involved.) Implementations for the FFT can be found in most numerical and signal processing libraries.

Now that we have defined the Fourier transform, what are some of its properties, and how can they be used? Table 3.1 lists a number of useful properties, which I describe in a little more detail below.

- **Superposition:** The Fourier transform of a sum of signals is the sum of their Fourier transforms. (Thus, the Fourier transform is a linear operator.)
- **Shift:** The Fourier transform of a shifted signal is the transform of the original signal times a *linear phase shift* (complex sinusoid).
- **Reversal:** The Fourier transform of a reversed signal is the complex conjugate of the signal's transform.
- **Convolution:** The Fourier transform of a pair of convolved signals is the product of their transforms.

Property	Signal	Transform
superposition	$f_1(x) + f_2(x)$	$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$	$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$	$F^*(\omega)$
convolution	$f(x) * h(x)$	$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$	$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$	$F(\omega) * H(\omega)$
differentiation	$f'(x)$	$j\omega F(\omega)$
domain scaling	$f(ax)$	$1/aF(\omega/a)$
real images	$f(x) = f^*(x) \Leftrightarrow F(\omega) = F(-\omega)$	
Parseval's Thm.	$\sum_x [f(x)]^2 = \sum_\omega [F(\omega)]^2$	

Table 3.1: *Some useful properties of Fourier transforms. The original transform pair is $F(\omega) = \mathcal{F}\{f(x)\}$.*

- **Correlation:** The Fourier transform of a correlation is the product of the first transform times the complex conjugate of the second one.
- **Multiplication:** The Fourier transform of the product of two signals is the convolution of their transforms.
- **Differentiation:** The Fourier transform of the derivative of a signal is that signal's transform multiplied by the frequency. In other words, differentiation linearly emphasizes (magnifies) higher frequencies.
- **Domain scaling:** The Fourier transform of a stretched signal is the equivalently compressed (and scaled) version of the original transform, and *vice versa*.
- **Real images:** The Fourier transform of a real-valued signal is symmetric around the origin. This fact can be used to save space and to double the speed of image FFTs by packing alternating scanlines into the real and imaginary parts of the signal being transformed.
- **Parseval's Theorem:** The energy (sum of squared values) of a signal is the same as the energy of its Fourier transform.

All of these properties are relatively straightforward to prove (see Exercise 3.15), and they will come in handy later on in the book, e.g., when designing optimum Wiener filters (§3.3.1) or performing fast image correlations (§7.1.2).

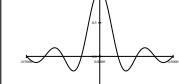
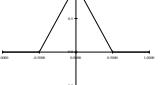
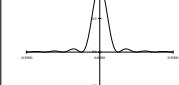
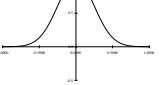
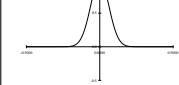
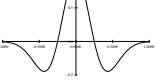
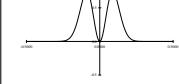
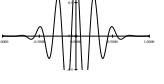
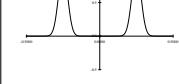
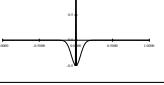
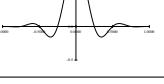
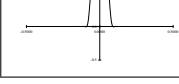
Name	Plot	Signal	Transform	Plot
impulse		$\delta(x)$	1	
shifted impulse		$\delta(x - u)$	$e^{-j\omega u}$	
box filter		$\text{box}(x/a)$	$a \text{sinc}(a\omega)$	
tent		$\text{tent}(x/a)$	$a \text{sinc}^2(a\omega)$	
Gaussian		$G(x; \sigma)$	$\frac{\sqrt{2\pi}}{\sigma} G(\omega; \sigma^{-1})$	
Lapl. of Gauss.		$(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})G(x; \sigma)$	$-\frac{\sqrt{2\pi}}{\sigma} \omega^2 G(\omega; \sigma^{-1})$	
Gabor		$\cos(\omega_0 x)G(x; \sigma)$	$\frac{\sqrt{2\pi}}{\sigma} G(\omega \pm \omega_0; \sigma^{-1})$	
unsharp mask		$(1 + \gamma)\delta(x) - \gamma G(x; \sigma)$	$(1 + \gamma) - \frac{\sqrt{2\pi}\gamma}{\sigma} G(\omega; \sigma^{-1})$	
windowed sinc		$r \cos(x/(aW)) \text{sinc}(x/a)$	(see Figure 3.30)	

Table 3.2: Some useful (continuous) Fourier transforms pairs. The dashed line in the Fourier transform of the shifted impulse indicates its (linear) phase. All other transforms have zero phase (they are real-valued). Note that the figures are not necessarily drawn to scale, but are rather drawn to illustrate the general shape and characteristics of the filter or its response. In particular, the Laplacian of a Gaussian is drawn inverted because it resembles more the “Mexican Hat” it is sometimes called.

Now that we have these properties in place, let us look at the Fourier transform pairs of some commonly occurring filters and signal, as listed in Table 3.2. In more detail, these pairs are as follows:

- **Impulse:** The impulse response has a constant (all frequency) transform.
- **Shifted impulse:** The shifted impulse has unit magnitude and linear phase.
- **Box filter:** The box (moving average) filter

$$\text{box}(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{else} \end{cases} \quad (3.54)$$

has a sinc Fourier transform,

$$\text{sinc}(\omega) = \frac{\sin \omega}{\omega}, \quad (3.55)$$

which has an infinite number of sidelobes. Conversely, the sinc filter is an ideal low-pass filter. For a non-unit box (Table 3.2), the width of the box a and the spacing of the zero crossings in the sinc $1/a$ are inversely proportional.

- **Tent:** The piecewise linear tent function,

$$\text{tent}(x) = \max(0, 1 - |x|), \quad (3.56)$$

has a sinc^2 Fourier transform.

- **Gaussian:** The (unit area) Gaussian of width σ ,

$$G(x; \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \quad (3.57)$$

has a (unit height) Gaussian of width σ^{-1} as its Fourier transform.

- **Laplacian of Gaussian:** The second derivative of a Gaussian of width σ ,

$$\text{LoG}(x; \sigma) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) G(x; \sigma) \quad (3.58)$$

has a band-pass response of

$$-\frac{\sqrt{2\pi}}{\sigma} \omega^2 G(\omega; \sigma^{-1}) \quad (3.59)$$

as its Fourier transform. [Note: Better check this with MatLab or Mathematica to be sure.

J

- **Gabor:** The even Gabor function, which is the product of a cosine of frequency ω_0 and a Gaussian of width σ , has as its transform the sum of the two Gaussians of width σ^{-1} centered at $\omega = \pm\omega_0$. The odd Gabor function, which uses a sine, is the difference of two such Gaussian. Gabor functions are often used for oriented and band-pass filtering, since they can be more frequency selective than Gaussian derivatives.
- **Unsharp mask:** The unsharp mask introduced in (3.22) has as its transform a unit response with a slight boost at higher frequencies.
- **Windowed sinc:** The windowed (masked) sinc function shown in Table 3.2 has a response function that approximates an ideal low-pass filter better and better as additional sidelobes are added (W is increased). Figure 3.30 shows the shapes of these such filters along with their Fourier transforms. For these examples, I use a one-lobe raised cosine,

$$\text{rcos}(x) = \frac{1}{2}(1 + \cos \pi x)\text{box}(x), \quad (3.60)$$

also known as the *Hann window* as the windowing function. [Wolberg \(1990\)](#) and [Oppenheim et al. \(1999\)](#) discuss additional windowing functions, which also include the *Lancosz* window, which is the positive first lobe of a sinc function.

We can also compute the Fourier transforms for the small discrete kernels shown in Figure 3.13, as shown in Table 3.3. Notice how the moving average filters do not uniformly dampen higher frequencies, and hence can lead to ringing artifacts. The binomial filter ([Gomes and Velho 1997](#)), used as the “Gaussian” in Burt and Adelson’s ([1983a](#)) Laplacian pyramid (§3.4), does a decent job of separating the high and low frequencies, but still leaves a fair amount of high-frequency detail, which can lead to aliasing after downsampling. The Sobel edge detector at first linearly accentuates frequencies, but then decays at higher frequency, and hence has trouble detecting fine-scale edges (e.g., adjacent black and white columns). We will see additional examples of small kernel Fourier transforms in §3.4.1, where we study better kernels for pre-filtering before decimation (size reduction).

Two-dimensional Fourier transforms

The formulas and insights we have developed for one-dimensional signals and their transforms translate directly to two-dimensional images. Here, instead of just specifying a horizontal or vertical frequency ω_x or ω_y , we can create an oriented sinusoid of frequency (ω_x, ω_y) ,

$$s(x, y) = \sin(\omega_x x + \omega_y y). \quad (3.61)$$

Name	Kernel	Transform	Plot
box-3	$\frac{1}{3} [1 \ 1 \ 1]$	$\frac{1}{3}(1 + 2 \cos \omega)$	
box-5	$\frac{1}{5} [1 \ 1 \ 1 \ 1 \ 1]$	$\frac{1}{5}(1 + 2 \cos \omega + 2 \cos 2\omega)$	
linear	$\frac{1}{4} [1 \ 2 \ 1]$	$\frac{1}{2}(1 + \cos \omega)$	
binomial	$\frac{1}{16} [1 \ 4 \ 6 \ 4 \ 1]$	$\frac{1}{4}(1 + \cos \omega)^2$	
Sobel	$\frac{1}{2} [-1 \ 0 \ 1]$	$\sin \omega$	
“Laplacian”	$\frac{1}{2} [-1 \ 2 \ -1]$	$\frac{1}{2}(1 - \cos \omega)$	

Table 3.3: Fourier transforms of the separable kernels shown in Figure 3.13.

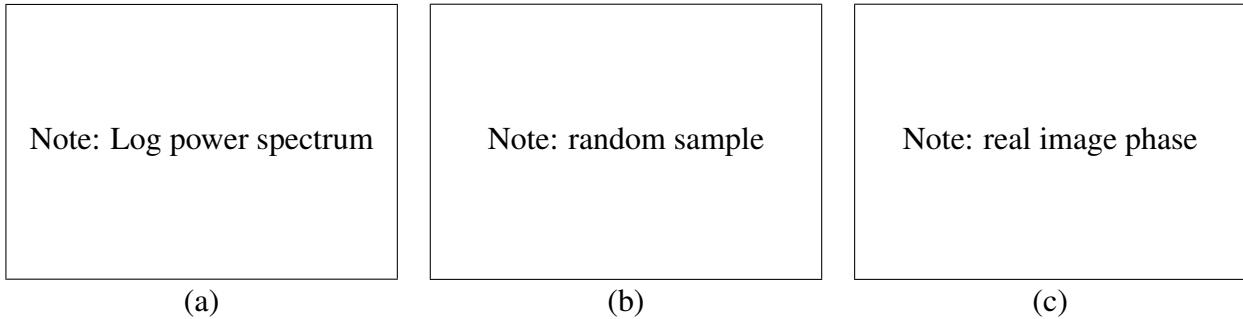


Figure 3.24: *Natural image power spectrum*: (a) typical power spectrum (log intensity); (b) random image sampled from this spectrum; (c) phase taken from a single image.
 [Note: Generate this figure]

The corresponding two-dimensional Fourier transforms are then

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} dx dy, \quad (3.62)$$

and in the discrete domain,

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j2\pi \frac{k_x x + k_y y}{MN}}, \quad (3.63)$$

where M and N are the width and height of the image.

All of the Fourier transform properties from Table 3.1 carry over to two dimensions if we replace the scalar variables x , ω , x_0 and a with their 2D vector counterparts $\mathbf{x} = (x, y)$, $\omega = (\omega_x, \omega_y)$, $\mathbf{x}_0 = (x_0, y_0)$, and $\mathbf{a} = (a_x, a_y)$, and use vector inner products instead of multiplications.

3.3.1 Wiener filtering

While the Fourier transform is a useful tool for analyzing the frequency characteristics of a filter kernel or image, it can also be used to analyze the frequency spectrum of a whole *class* of images. Consider, for example, a random selection of 100 photographs from your personal photo collection (Exercise 3.16). If you take their Fourier transform and average the squared magnitude of their transforms, you get an image that looks something like Figure 3.24a.

A simple model for images would be to assume that they are random noise fields whose expected magnitude at each frequency is given by this *power spectrum* $P_s(\omega_x, \omega_y)$, i.e.,

$$\langle [S(\omega_x, \omega_y)]^2 \rangle = P_s(\omega_x, \omega_y), \quad (3.64)$$

where the angle brackets $\langle \cdot \rangle$ denote the expected (mean) value of a random variable. (The notation $E[\cdot]$ is also often used.) [Note: Decide which to use in this book.] To generate such an image,

we simply create a random Gaussian noise image $S(\omega_x, \omega_y)$ where each “pixel” is a zero-mean⁷ Gaussian⁸ of variance $P_s(\omega_x, \omega_y)$ and then take its inverse FFT.

Figure 3.24b shows such a typical image, which, unfortunately, looks more like a [Note: *fill this in*] than a real image. It is well known that the phase of an image’s Fourier transform carries more information than its magnitude. Figure 3.24c shows what happens if we use the average power spectrum combined with a particular image’s phase to regenerate that image.

The observation that signal spectra capture a first order description of spatial statistics is widely used in signal and image processing. In particular, assuming that an image is a sample from a correlated Gaussian random noise field combined with a statistical model of the measurement process yields an optimum restoration filter known as the *Wiener filter*.⁹

To derive the Wiener filter, we analyze each frequency component of a signal’s Fourier transform independently. The noisy image formation process can be written as

$$o(x, y) = s(x, y) + n(x, y), \quad (3.65)$$

where $s(x, y)$ is then (unknown) image we are trying to recover, $n(x, y)$ is the additive noise signal, and $o(x, y)$ is the *observed* noisy image. Because of the linearity of the Fourier transform, we can equivalently write

$$O(\omega_x, \omega_y) = S(\omega_x, \omega_y) + N(\omega_x, \omega_y), \quad (3.66)$$

where each quantity in the above equation is the Fourier transform of the corresponding image.

At each frequency (ω_x, ω_y) , we know from our image spectrum that the unknown transform component $S(\omega_x, \omega_y)$ has a *prior* distribution which is a zero-mean Gaussian with variance $P_s(\omega_x, \omega_y)$. We also have noisy measurement $O(\omega_x, \omega_y)$ whose variance is $P_n(\omega_x, \omega_y)$, i.e., the power spectrum of the noise, which is usually assumed to be constant (white), $P_n(\omega_x, \omega_y) = \sigma_n^2$.

According to Bayes’ Rule (Appendix B.4), [Note: *fix this reference later*], the *posterior estimate* of S can be written as

$$p(S|O) = \frac{p(O|S)p(S)}{p(O)}, \quad (3.67)$$

where $p(O) = \int_S p(O|S)p(S)$ is a normalizing constant used to make the $p(S|O)$ distribution *proper* (integrate to 1). The prior distribution $P(S)$ is given by

$$p(S) = e^{-\frac{(S-\mu)^2}{2P_s}}, \quad (3.68)$$

⁷ Except for the DC component at $(\omega_x, \omega_y) = (0, 0)$, which is set to mean gray.

⁸ See §C.2 for code to generate Gaussian noise. [Note: Replace with a more detailed reference, e.g., equation number, when that section is finished.]

⁹ Wiener is pronounced “veener”, since in German, the “w” is pronounced “v” and the “v” is pronounced “f”. Remember that next time you order “Wiener schnitzel”.

where μ is the expected mean at that frequency (0 everywhere except at the origin), and the measurement distribution $P(O|S)$ is given by

$$p(S) = e^{-\frac{(S-O)^2}{2P_n}}. \quad (3.69)$$

Taking the negative logarithm of both sides of (3.67) and setting $\mu = 0$ for simplicity, we get

$$-\log p(S|O) = -\log p(O|S) - \log p(S) + C \quad (3.70)$$

$$= 1/2P_n^{-1}(S - O)^2 + 1/2P_s^{-1}S^2 + C, \quad (3.71)$$

which is the *negative posterior log likelihood*. The minimum of this quantity is easy to compute,

$$S_{\text{opt}} = \frac{P_n^{-1}}{P_n^{-1} + P_s^{-1}}O = \frac{P_s}{P_s + P_n}O = \frac{1}{1 + P_n/P_s}O. \quad (3.72)$$

The quantity

$$W(\omega_x, \omega_y) = \frac{1}{1 + \sigma_n^2/P_s(\omega_x, \omega_y)} \quad (3.73)$$

is the Fourier transform of the optimum *Wiener filter* needed to remove the noise from an image whose power spectrum is $P_s(\omega_x, \omega_y)$.

Notice that this filter has the right qualitative properties, i.e., for low frequencies where $P_s \gg \sigma_n^2$ it has unit gain, whereas for high frequencies it attenuates the noise by a factor P_s/σ_n^2 . Figure 3.25 shows the one-dimensional transform $W(f)$ and the corresponding filter kernel $w(x)$ for the commonly assumed case of $P(f) = f^{-1}$, [Note: check that this exponent is correct!] while Figure 3.26 shows a 2D image denoising example using the empirically estimated power spectrum shown in Figure 3.24a. Exercise 3.16 has you compare the Wiener filter as a de-noising algorithm to hand-tuned Gaussian smoothing.

The methodology given above to derive the Wiener filter can easily be extended to the case where the observed image is a noisy blurred version of the original image,

$$o(x, y) = b(x, y) * s(x, y) + n(x, y), \quad (3.74)$$

where $b(x, y)$ is the known blur kernel. Rather than deriving the corresponding Wiener filter, I leave it as an exercise (Exercise 3.17), which also encourages you to compare your de-blurring results with unsharp masking and naïve inverse filtering. More sophisticated algorithms for blur removal will be discussed in §3.6 and §9.3.

[Note: Show the complete noise/ σ_b table suggested in Exercise 3.16? Also, show de-blur results here? Let's see if I have space and time (good example for class slides).]

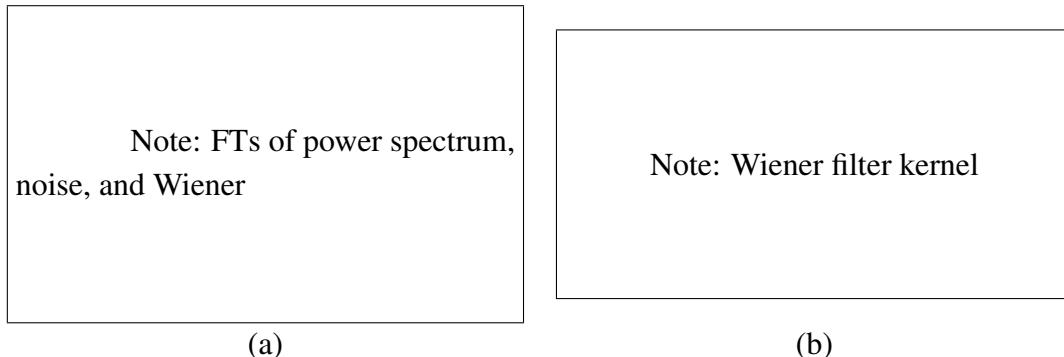


Figure 3.25: One-dimensional Wiener filter: (a) power spectrum of signal $P_s(f)$, noise level σ^2 , and Wiener filter transform $W(f)$; (b) Wiener filter spatial kernel.

[Note: Generate this figure]

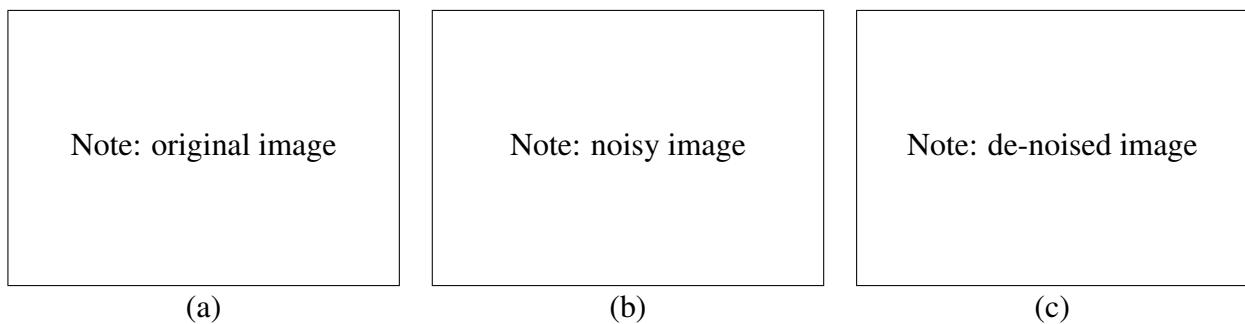


Figure 3.26: Wiener filtering example: (a) original image; (b) noisy image; (c) de-noised image.

[Note: Generate this figure]

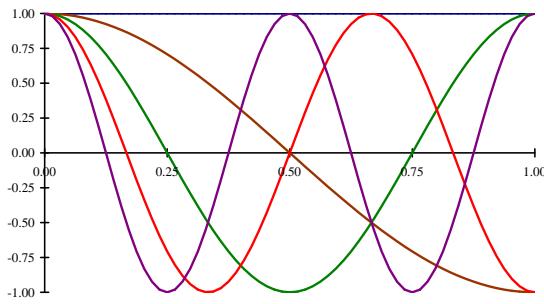


Figure 3.27: Discrete cosine transform (DCT) basis functions. The first (DC) basis is the horizontal blue line, the second is the brown half-cycle waveform, etc. These bases are widely used in image and video compression standards such as JPEG.

Discrete cosine transform

The *discrete cosine transform* (DCT) is a variant of the Fourier transform particularly well suited to compressing images in a block-wise fashion. The one-dimensional DCT is computed by taking the dot product of each N -wide block of pixels with a set of cosines of different frequencies,

$$F(k) = \sum_{i=0}^{N-1} \cos\left(\frac{\pi}{N}(i + \frac{1}{2})k\right) f(i), \quad (3.75)$$

where k is the coefficient (frequency) index, and the $1/2$ -pixel offset is used to make the basis coefficients symmetric in time (Wallace 1991). Some of the discrete cosine basis functions are shown in Figure 3.27. As you can see, the first basis function (straight blue line) encodes the average DC value in the block of pixels, while the second encodes (a slightly curvy version of) the slope.

It turns out that the DCT is a good approximation to the optimal Karhunen-Loëve decomposition of natural image statistics over small patches, which can be obtained by performing a principal component analysis (PCA) of images, as described in §14.1.1. The KL-transform de-correlates the signal optimally (assuming the signal is described by its spectrum), and thus theoretically leads to optimal compression.

The two-dimensional version of the DCT is defined similarly,

$$F(k, l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \cos\left(\frac{\pi}{N}(i + \frac{1}{2})k\right) \cos\left(\frac{\pi}{N}(j + \frac{1}{2})l\right) f(i, j). \quad (3.76)$$

Like the 2D Fast Fourier Transform, the 2D DCT can be implemented separably, i.e., first computing the DCT of each line in the block, and then computing the DCT of each resulting column. Like the FFT, each of the DCTs can be computed in $O(N \log N)$ time, using a simple adaptation of standard FFT algorithms to the special case of real-valued input coefficients.

As I mentioned in §2.3.3, the DCT is widely used in today's image and video compression algorithms, although it is slowly being supplanted by wavelet algorithms §3.4.3 (Simoncelli and Adelson 1990b) and overlapped variants of the DCT (Malvar 1990, Malvar 1998, Malvar 2000). These newer algorithms suffer less from the *blocking artifacts* (visible edge-aligned discontinuities) that result from the pixels in each block (typically 8×8) being transformed and quantized independently. (See Exercise 3.30 for ideas on how to remove blocking artifacts from compressed JPEG images.)

3.3.2 Application: Sharpening, blur, and noise removal

[Note: Still need to write this section.]

These are simple operations, which can change the appearance, and also remove noise.

Review both linear and non-linear variants, e.g., Gaussian, median, and bi-lateral, as well as Wiener filtering.

Show some examples (refer back to Figure 3.18) and maybe show a few more.

Refer students to exercise Ex 3.11, where they are encouraged to take blurry or noisy images (shooting in low light is a good way to get both) and to try to improve their appearance and legibility.

Mention how GPUs make it easier/faster to do some of these operations.

3.4 Pyramids and wavelets

So far in this chapter, all of the image transformations we have studied produce output images of the same size as the inputs. Often, however, we may wish to change the resolution of an image before proceeding further. For example, we may need to interpolate a small image to make its resolution match that of the output printer or computer screen. Alternatively, we may want to reduce the size of an image to speed up the execution of an algorithm or to save on storage space or transmission time.

Sometimes, we do not even know what the appropriate resolution for the image should be. Consider, for example, the task of finding a face in an image (§14.2). Since we do not know at what scale the face will appear, we need to generate a whole *pyramid* of differently sized images and scan each one for possible faces. (Biological visual systems also operate at a hierarchy of scales (Marr 1982).) Such a pyramid can also be very helpful in *accelerating* the search for an object by first finding a smaller instance of that object at a smaller (coarser) level of the pyramid and then looking for the full resolution object only in the vicinity of coarse-level detections (§7.1.1). Finally, image pyramids are extremely useful for performing multi-scale editing operations such as *blending* images while maintaining details.

In this section, I first discuss good filters for changing image resolutions (upsampling and downsampling, *aka* interpolation and decimation), §3.4.1. I then present the concept of multi-resolution pyramids, which can be used to create a complete hierarchy of different sized images and which enable a variety of applications, §3.4.2. A closely related concept is that of *wavelets*, which are a special kind of pyramid with higher frequency selectivity and other useful properties, §3.4.3. Finally, I present a useful application of pyramids, namely the blending of different images in a way that hides the seams between the image boundaries, §3.4.4.

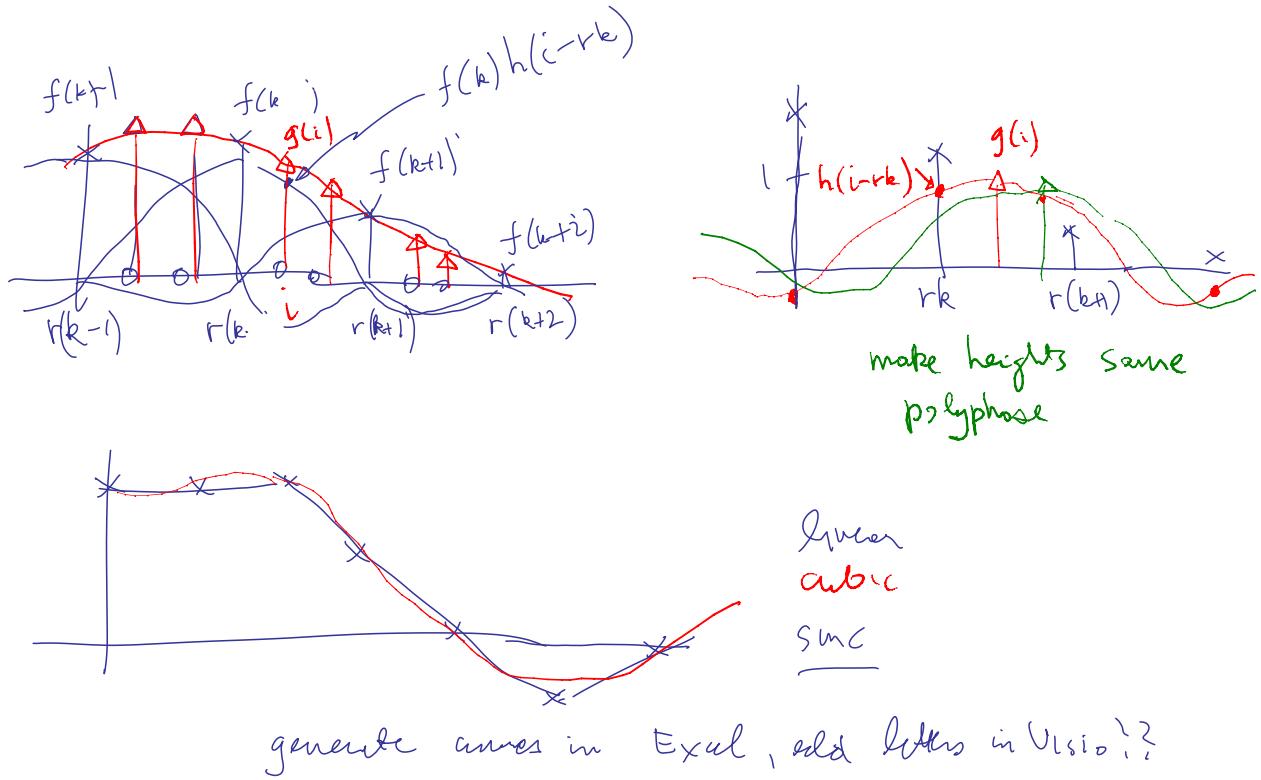


Figure 3.28: *Signal interpolation: (a) weighted summation of input values; (b) polyphase filter interpretation; (c) sample interpolants (linear, cubic, and sinc).*

[Note: Generate the curves in Excel and add the labels in Visio?]

3.4.1 Interpolation and decimation

Before we can construct and image pyramid, we first need to define how images can be interpolated (made larger) and decimated (made smaller).

Interpolation

In order to *interpolate* (or *upsample*) an image to a higher resolution, we need to select some interpolation *kernel* with which to convolve the image,

$$g(i, j) = \sum_{k,l} f(k, l)h(i - rk, j - rl). \quad (3.77)$$

This formula is related to the discrete convolution formula (3.14) except that we replace k and l inside of $h()$ with rk and rl , where r is the upsampling rate. Figure 3.28a shows how to think of this process as the superposition of sample weighted interpolation kernels, one centered at each input sample k . An alternative mental model is shown in Figure 3.28b, where the kernel is centered

at the output pixel value i . (The two forms are equivalent.) The latter form is sometimes called the *polyphase filter* form, since the kernel values $h(i)$ (in the one-dimensional separable case) can be stored as r separate kernels, each of which is selected for convolution with the input samples depending on the *phase* of i relative to the upsampled grid.

What kinds of kernels make good interpolators? The answer depends on the application and the computation time involved. Any of the smoothing kernels shown in Tables 3.2 and 3.3 can be used after appropriate re-scaling.¹⁰ The *linear* interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves (Figure 3.28c), which result in unappealing *creases* when applied to images (Figure 3.29a). The cubic B-spline, whose discrete $1/2$ -pixel sampling appears as the *binomial kernel* in Table 3.3, is an *approximating* kernel (the interpolated image does not pass through the input data points) that produces soft images with reduced high-frequency detail. The equation for the cubic B-spline is easiest to derive by convolving the tent function (linear B-spline) with itself.

While most graphics cards use the bilinear kernel (optionally combined with a MIP-map §3.4.2), most photo editing packages use *bicubic* interpolation. [Note: Rationalize bi[-]linear and bi[-]cubic. What do the graphics textbooks (Foley et al. 1995) and spline books (Farin 1992) do?] The cubic interpolant is a C^1 (derivative-continuous) piecewise-cubic *spline* (the term spline¹¹ and piecewise-polynomial are synonymous) whose equation is

$$h(x) = \begin{cases} 1 - (a + 3)x^2 + (a + 2)|x|^3 & \text{if } |x| < 1 \\ a(|x| - 1)(|x| - 2)^2 & \text{if } 1 \leq |x| < 2 \\ 0 & \text{else} \end{cases}, \quad (3.78)$$

where a specifies the derivative at $x = 1$ (Parker *et al.* 1983). The value of a is often set to $a = -1$, since this best matches the frequency characteristics of a sinc function (Figure 3.30). It also introduces a small amount of sharpening, which can be visually appealing. Unfortunately, this choice does not linearly interpolate straight lines (intensity ramps), so some visible ringing may occur. A better choice for large amounts of interpolation is probably $a = -0.5$, which produces a *quadratic reproducing* spline (it interpolates linear and quadratic functions exactly) (Wolberg 1990, §5.4.3). [Note: In our VisionTools library, the default value of $a = -1$. We may want to change this for large amounts of interpolation. See what GDI+ and Photoshop do?] Figure 3.30 shows the $a = -1$ and $a = -0.5$ cubic interpolating kernel along with their Fourier transforms; Figures 3.28c and 3.29b show them being applied to one- and two-dimensional interpolation.

Splines have long been used for function and data value interpolation because of the ability to

¹⁰ The smoothing kernels in Table 3.3 have a unit area. To turn these into interpolating kernels, we simply scale them up by the interpolation rate r .

¹¹ The origin of the term spline comes from the draughtsman's workshop, where it was the name of a flexible piece of wood or metal used to draw smooth curves.

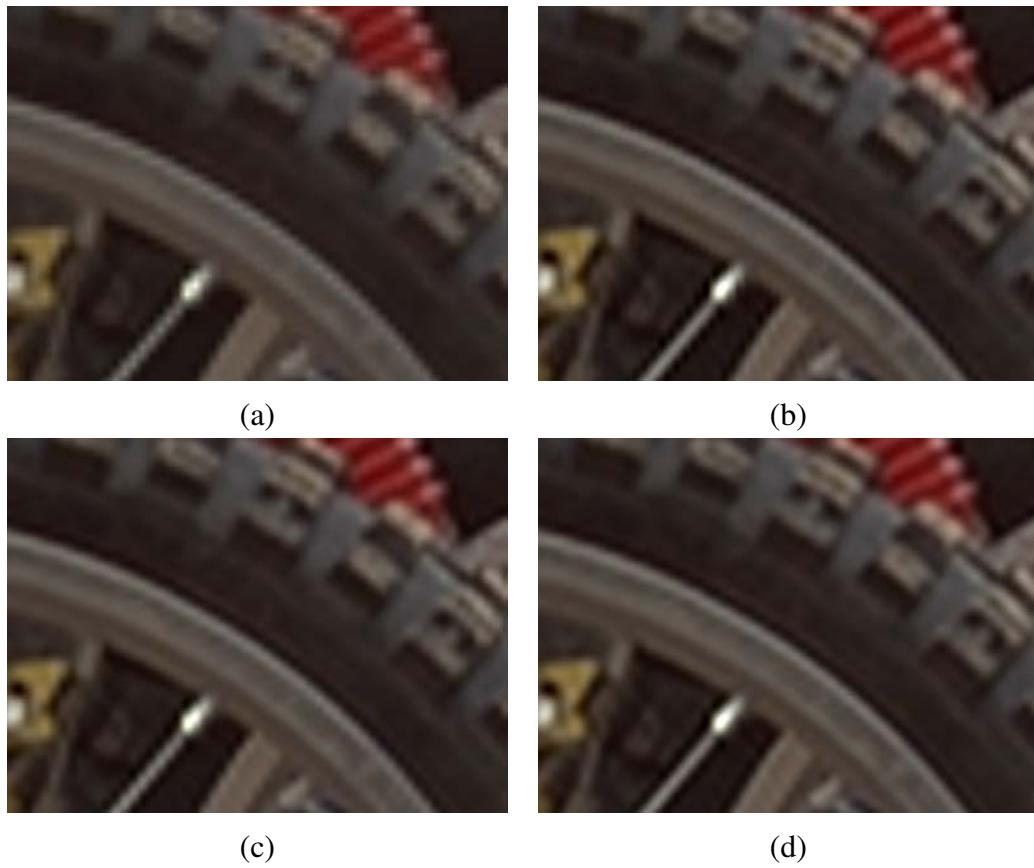


Figure 3.29: Two-dimensional image interpolation: (a) bilinear; (b) bicubic ($a = -1$); (c) bicubic ($a = -0.5$); (d) windowed sinc ([Note: ?] taps).

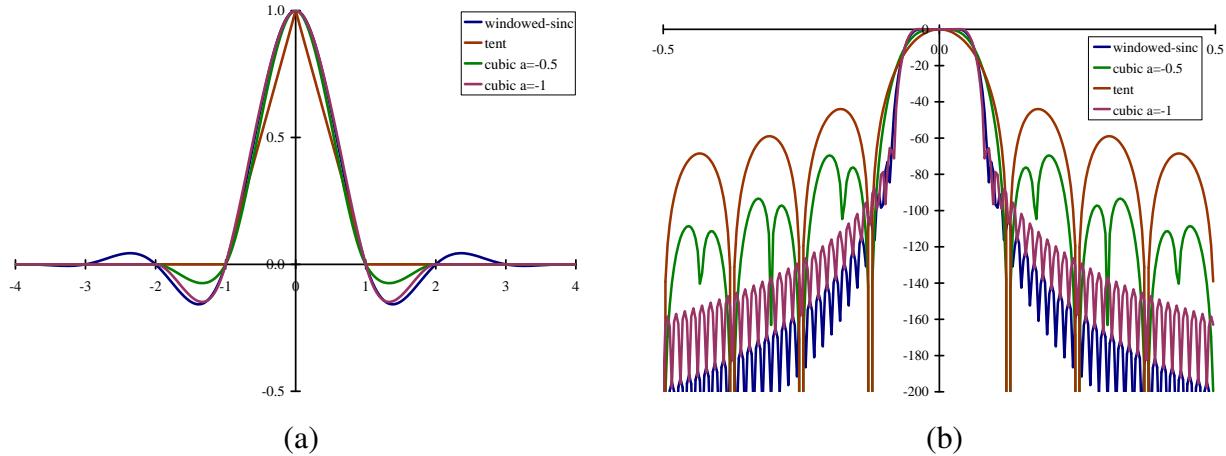


Figure 3.30: Some windowed sinc functions (a) and their log Fourier transforms (b): raised-cosine windowed sinc in blue, cubic interpolators ($a = -1$ and $a = -0.5$) in green and purple, and tent function in brown. These are often used to perform high-accuracy low-pass filtering operations.

precisely specify derivatives at control points and efficient *incremental* algorithms for their evaluation (Bartels *et al.* 1987). Splines are widely used in geometric modeling and computer-aided design (CAD) applications, although they have started being displaced by subdivision surfaces [Note: need reference here]. In computer vision, splines are often used for elastic image deformations §3.5.2, motion estimation §7.3, and surface interpolation §11.5. In fact, it is possible to carry out most image processing operations by representing images as splines and manipulating them in a multi-resolution framework (Unser 1999).

The highest quality interpolator is generally believed to be the windowed sinc function, because of it both preserves details in the lower resolution image and avoids aliasing. (It is also possible to construct a C^1 piecewise-cubic approximation to the windowed sinc by matching its derivatives at zero crossing (Szeliski and Ito 1986).) However, some people object to the excessive *ringing* that can be introduced by the windowed sinc and to the repetitive nature of the ringing frequencies (see Figures 3.28c and 3.29d). For this reason, some photographers prefer to repeatedly interpolate images by a small fractional amount (this tends to de-correlate the original pixel grid with the final image). [Note: Make this an exercise?] Additional possibilities include using the bilateral filter as an interpolator (Kopf *et al.* 2007a), using global optimization §3.5, or hallucinating details §9.3.

Decimation

While interpolation can be used to increase the resolution of an image, decimation (downsampling) is required to reduce the resolution.¹² To perform decimation, we first (conceptually) convolve the image with a low-pass filter (to avoid aliasing) and then keep every r th sample. In practice, we usually only evaluate the convolution at every r th sample,

$$g(i, j) = \sum_{k,l} f(k, l)h(ri - k, rj - l), \quad (3.79)$$

as shown in Figure 3.31. Note that the smoothing kernel $h(k, l)$ in this case is often a stretched and re-scaled version of an interpolation kernel. Alternatively, we can write

$$g(i, j) = \frac{1}{r} \sum_{k,l} f(k, l)h(i - k/r, j - l/r) \quad (3.80)$$

and keep the same kernel $h(k, l)$ for both interpolation and decimation.

One commonly used ($r = 2$) decimation filter is the *binomial* filter introduced by Burt and Adelson (1983a). As shown in Figure 3.3, this kernel does a decent job of separating the high and low frequencies, but still leaves a fair amount of high-frequency detail, which can lead to

¹² The term *decimation* has a gruesome etymology relating to practice of killing every 10th soldier in a platoon for failing to meet objectives. It is generally used in signal processing for any downsampling or rate reduction.

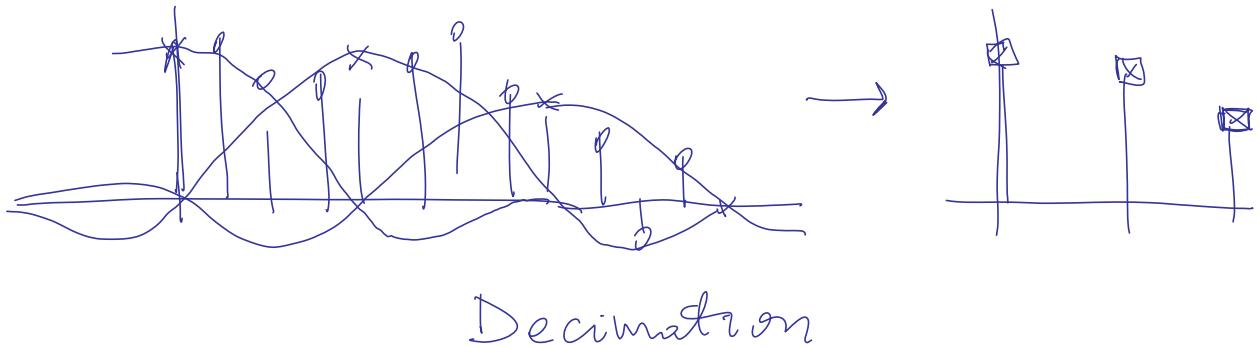


Figure 3.31: *Signal decimation: the original samples are convolved with a low-pass filter before being downsampled.*

$ n $	Linear	Binomial	Cubic $a = -1$	Cubic $a = -0.5$	Wind. sinc	QMF-9	JPEG 2000	Photoshop
0	0.50	0.3750	0.5000	0.50000	0.4939	0.5638	0.6029	
1	0.25	0.2500	0.3125	0.28125	0.2684	0.2932	0.2669	
2		0.0625	0.0000	0.00000	0.0000	-0.0519	-0.0782	
3			-0.0625	-0.03125	-0.0153	-0.0431	-0.0169	
4					0.0000	0.0198	0.0267	

Table 3.4: *Filter coefficients for $2 \times$ decimation. These filters are of odd length and symmetric and are normalized to have unit DC gain (sum up to 1). See Figure 3.32 for their associated frequency responses.*

[Note: Get the Photoshop result and decide whether to include. If so, also refresh the plot.]

aliasing after downsampling. However, for application such as image blending (discussed later in this section), this aliasing is of little concern.

If, however, the downsampled images will be displayed directly to the user, or perhaps blended with other resolutions as in MIP-mapping §3.4.2, a higher-quality filter is desired. For high down-sampling rates, the windowed sinc pre-filter is a good choice (Figure 3.30). However, for small down-sampling rates, e.g., $r = 2$, more careful filter design is required.

Table 3.4 shows a number of commonly used $r = 2$ down-sampling filters, while Figure 3.32 shows their corresponding frequency responses. These filters include:

- the linear [1, 2, 1] filter: relatively poor response;
- the binomial [1, 4, 6, 4, 1] filter: cuts off a lot of frequencies, but useful for computer vision analysis pyramids;

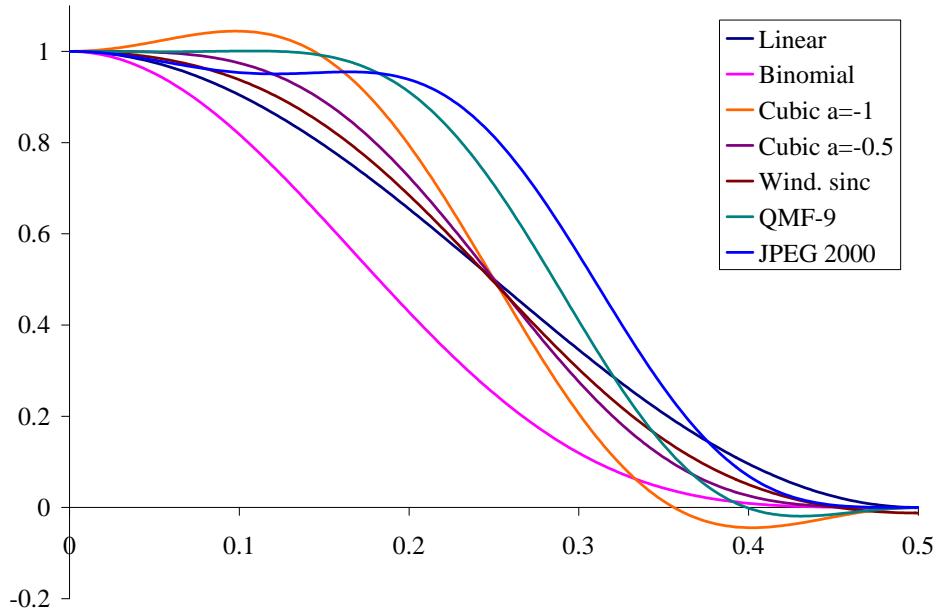


Figure 3.32: Frequency response for some $2 \times$ decimation filters. The cubic $a = -1$ filter has the sharpest fall-off but also a bit of ringing; the wavelet analysis filters (QMF-9 and JPEG 2000), while useful for compression, have more aliasing.

- the cubic filters from (3.78): the $a = -1$ filter has a sharper fall-off than the $a = -0.5$ filter (Figure 3.32);
- a cosine-windowed sinc function (Table 3.2);
- the QMF-9 filter of Simoncelli and Adelson (1990b): used for wavelet denoising, aliases a fair amount (note that their original filter coefficients are normalized to $\sqrt{2}$ gain so they can be “self-inverting”);
- the 9/7 analysis filter from JPEG 2000 (Taubman and Marcellin 2002)

Please see the original papers for the full-precision values of some of these coefficients.

[Note: Show the frequency response of some small kernels (use 9-tap from VisionTools (does not seem to exist! try to find Rico’s e-mail?))]

[Note: Show some images decimated by 2 with some different filters, and give a pointer to the on-line supplementary materials, where you can see these better.]

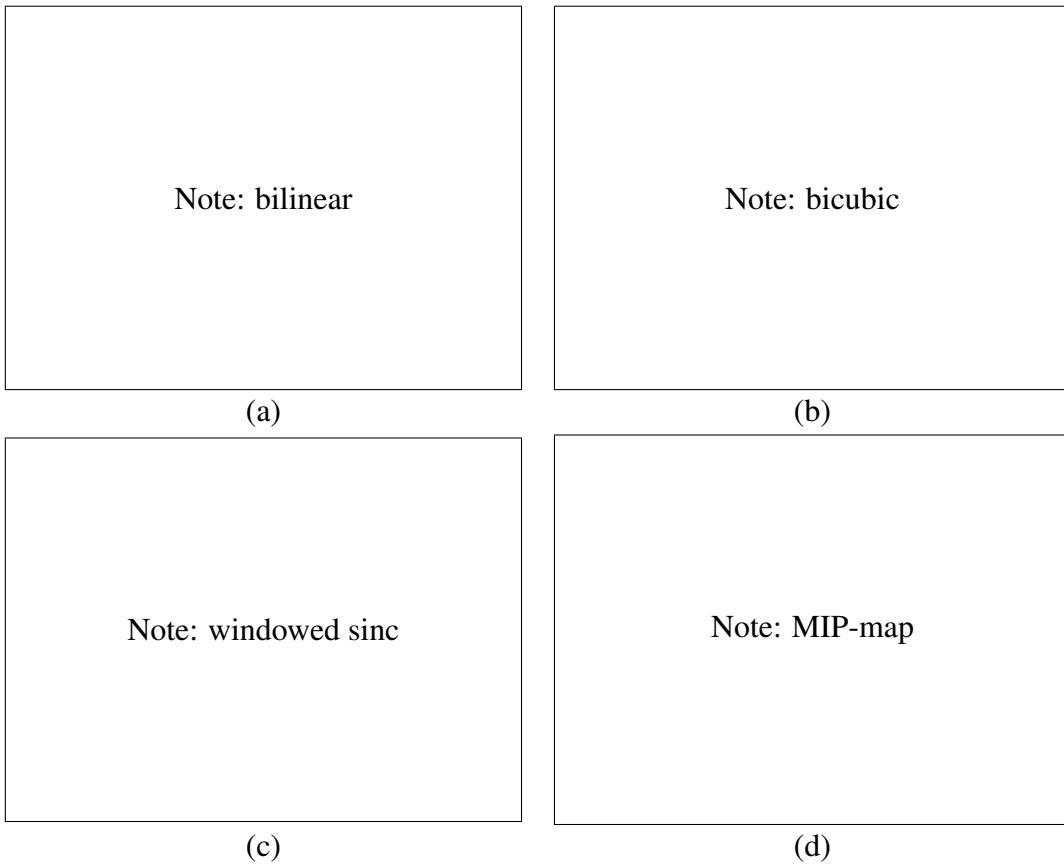


Figure 3.33: *Two-dimensional image decimation: (a) bilinear; (b) bicubic ($a = [\text{Note: ?}]$); (c) windowed sinc ($[\text{Note: ?}]$ taps); (d) tri-linear MIP-map.*

[Note: Choose an half-octave interpolation ratio to better see MIP-map artifacts.]

3.4.2 Multi-resolution representations

Now that I have described interpolation and decimation algorithms, we can build a complete image pyramid (Figure 3.34). As I mentioned before, pyramids can be used to accelerate coarse-to-fine search algorithms, look for objects or patterns at different scales, and to perform multi-resolution blending operations. They are also widely used in computer graphics hardware and software to perform fractional-level smoothing using the MIP-map, which I cover in §3.5.

The best known (and probably most widely used) pyramid in computer vision is Burt and Adelson's (1983a) Laplacian pyramid. To construct the pyramid, we first blur and subsample the original image by a factor of two and store this in the next level of the pyramid (Figure 3.35). Because adjacent levels in the pyramid are related by a sampling rate $r = 2$, this kind of pyramid is known as an *octave pyramid*. In Burt and Adelson's work, they originally propose a 5-tap kernel

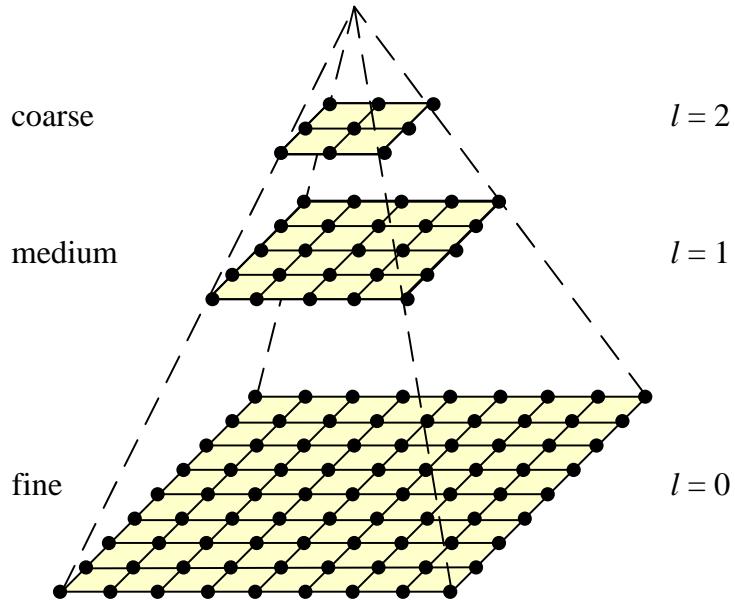


Figure 3.34: A traditional image pyramid: each level has half the resolution (width and height) and hence a quarter of the pixels as its parent level.

of the form

$$\boxed{c \ b \ a \ b \ c}, \quad (3.81)$$

with $b = 1/4$ and $c = 1/4 - a/2$. In practice, they and everyone else uses $a = 3/8$, which results in the familiar binomial kernel,

$$\frac{1}{16} \boxed{1 \ 4 \ 6 \ 4 \ 1}, \quad (3.82)$$

which is particularly easy to implement using shifts and adds. (This was important in the days when multipliers were expensive.) The reason they call their resulting pyramid a *Gaussian* pyramid is that repeated convolutions of the binomial kernel converge to a Gaussian. (But then again, this is true for any smoothing kernel (Wells 1986).) [Note: Re-read (Burt 1981) and (Wells 1986) and see who did what. Note that (Wells 1986) is NOT in IEEE Explore (too old) or on Sandy's Web page, so will need to find hardcopy.]

[Note: Need an actual figure here (similar to Figure 3.41) showing a Gaussian and Laplacian pyramid decomposition of an image. Use one of my personal photos, e.g., the Iris?]

To compute the *Laplacian* pyramid, Burt and Adelson first interpolate a lower resolution image to obtain a *reconstructed* low-pass version of the original image (Figure 3.36b). They then subtract this low-pass version from the original to yield the band-pass “Laplacian” image, which can be stored away for further processing. The resulting pyramid is *self-inverting*, i.e., the Laplacian images plus the base-level Gaussian (L_2 in Figure 3.36b) are sufficient to exactly reconstruct the original image. Figure 3.35 shows the same computation in one dimension as a *signal processing*

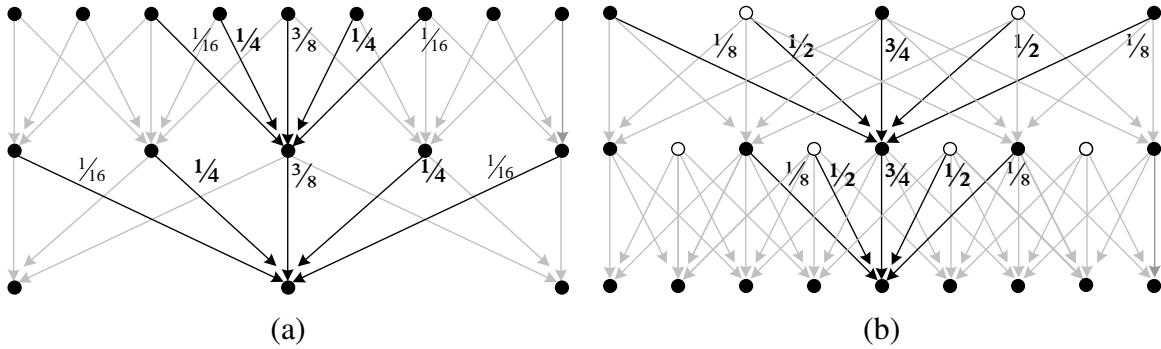


Figure 3.35: The Gaussian pyramid shown as a signal processing diagram. The analysis (a) and (re-)synthesis (b) stages are shown as using similar computations. The white circles indicate zero values inserted by the $\uparrow 2$ upsampling operation. Notice how the reconstruction filter coefficients are twice the analysis coefficients. The computation is shown as flowing down the page, regardless of whether we are going coarse to fine or vice versa.

[Note: Crop in Acrobat white margins does not work for the 2nd figure, so have to use manual cropping with the crop tool (then double click inside). See Acrobat Crop pages help page under Crop a page with the crop tool.]

diagram, which completely captures the computations being performed during the analysis and re-synthesis stages.

Burt and Adelson also describe a variant on the Laplacian pyramid where the low-pass image is taken from the original blurred image rather than the reconstructed pyramid (piping the output of the L box directly to the subtraction in Figure 3.36b). This variant has less aliasing, since it avoids one downsampling and upsampling round-trip, but it is not self-inverting, since the Laplacian images are no longer adequate to reproduce the original image.

As before with the Gaussian pyramid, the term Laplacian is a bit of a misnomer, since their band-pass images are really differences of (approximate) Gaussians, or DoGs,

$$\text{DoG}\{I; \sigma_1, \sigma_2\} = G_{\sigma_1} * I - G_{\sigma_2} * I = (G_{\sigma_1} - G_{\sigma_2}) * I. \quad (3.83)$$

A Laplacian of a Gaussian (which we saw previously in (3.26)) is actually its second derivative,

$$\text{LoG}\{I; \sigma\} = \nabla^2(G_\sigma * I) = (\nabla^2 G_\sigma) * I, \quad (3.84)$$

where

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad (3.85)$$

is the Laplacian (operator) of a function. Figure 3.37 shows how the Difference of Gaussians and Laplacian of Gaussian look like in both space and frequency.

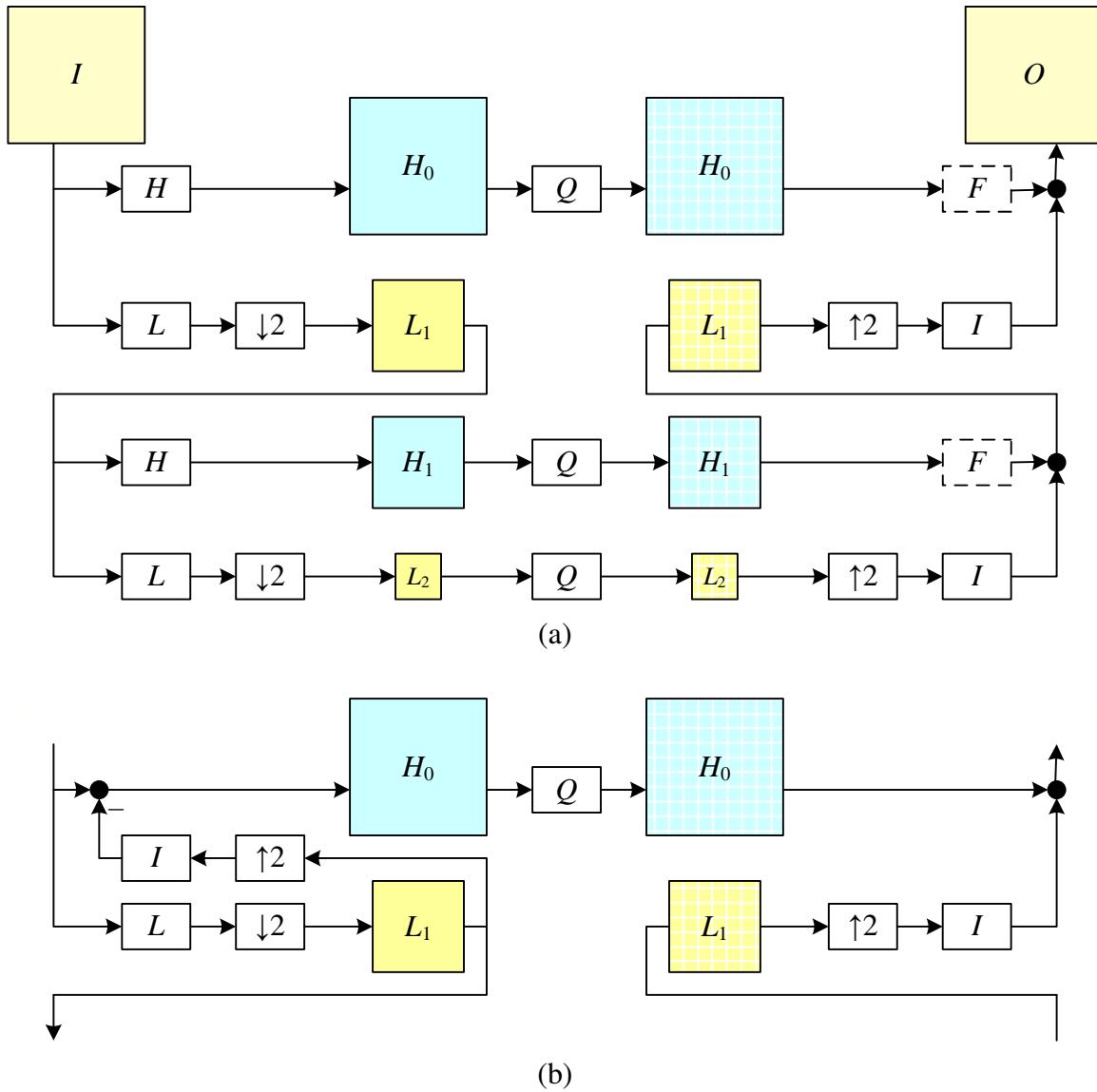


Figure 3.36: *The Laplacian pyramid.* (a) *The conceptual flow of images through processing stages: images are high-pass and low-pass filtered, and the low-pass filtered images are processed in the next stage of the pyramid. During reconstruction, the interpolated image and the (optionally filtered) high-pass image are added back together. The Q box indicates quantization or some other pyramid processing (e.g., noise removal by coring).* (b) *The actual computation of the high-pass filter involves first interpolating the downsampled low-pass image and then subtracting it. This results in perfect reconstruction when Q is the identity. The high-pass (or band-pass) images are typically called Laplacian images, while the low-pass images are called Gaussian images.*

[Note: Consider adding images where the colored boxes are.]

Band-pass Filter = Difference of Low-Pass Filters

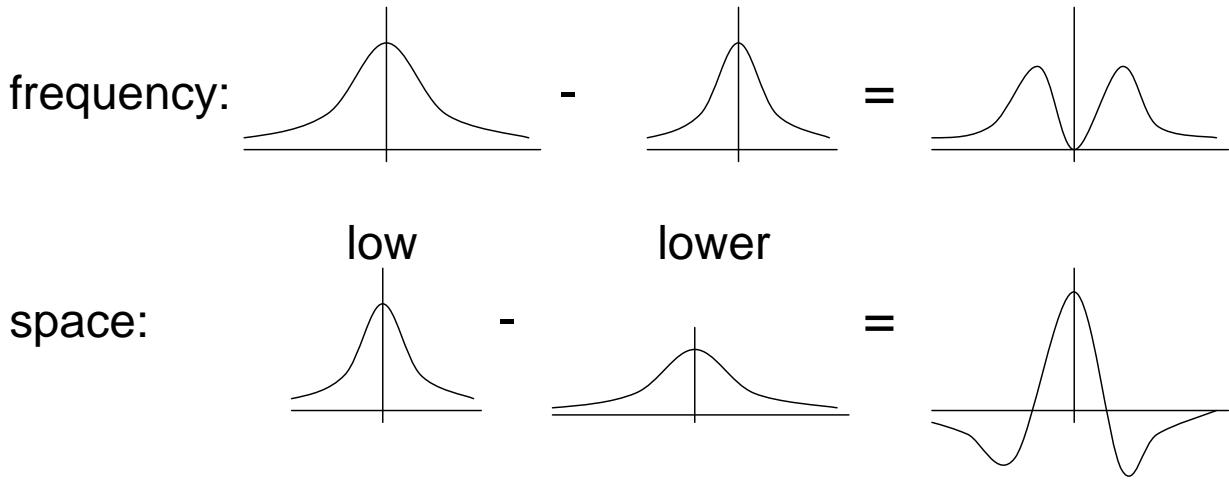


Figure 3.37: The difference of two low-pass filters results in a band-pass filter.

[Note: Fix up this figure and consider whether to show the difference between DoG and LoG. Could even include the Burt-Adelson 5-tap version, although because of aliasing, the filter and response are shift-variant.]

Laplacians of Gaussian have elegant mathematical properties that have been widely studied in the *scale space* community (Witkin 1983, Witkin *et al.* 1986, Lindeberg 1990, Nielsen *et al.* 1997) and can be used for a variety of applications including edge detection (Perona and Malik 1990b), stereo matching (Witkin *et al.* 1987), and image enhancement (Nielsen *et al.* 1997).

A less widely used variant of pyramids are *half-octave pyramids*, shown in Figure 3.38. These were first introduced to the vision community by Crowley and Stern (1984), who call them *Difference of Low-Pass* (DOLP) transforms. Because of the small scale change between adjacent levels, the authors claim that coarse-to-fine algorithms perform better. In the image-processing community, half-octave pyramids combined with the checkerboard sampling grids are known as *quincunx* sampling (Feilner *et al.* 2005). In detecting multi-scale features §4.1.1, it is often common to use half-octave or even quarter-octave pyramids (Lowe 2004, Triggs 2004). However, in this case, the subsampling only occurs at every octave level, i.e., the image is repeatedly blurred with wider Gaussians until a full octave of resolution change has been achieved. [Note: Is there a picture in Trigg's paper (Triggs-ECCV04.pdf), or a better description? I did not find one (maybe Bill can help). Should I include such a diagram here or in the features section?]

[Note: Mention high-quality decimation (7/9 filter; see wavelets below)]

[Note: Need to finish off this paragraph on non-linear filtering:]

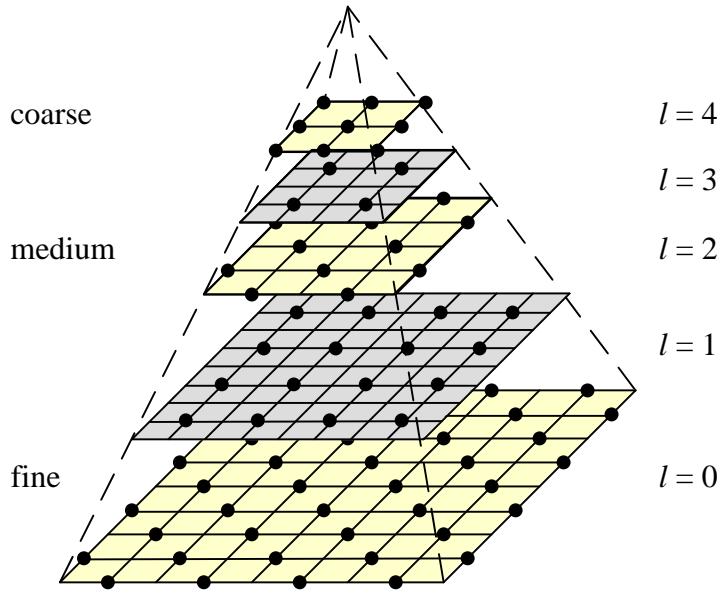


Figure 3.38: *Multiresolution pyramid with half-octave (quincunx) sampling (odd levels are colored gray for easier visibility).*

Since non-linear filtering often outperforms linear filtering, one might ask whether people have tried constructing pyramids using non-linear operators? In fact, they recently have. Gluckman (2006a, 2006b) [Note: describe what he does here] In computer graphics... Non-linear pyramids: talk about Gluckman (2006a)'s Higher Order Pyramids and Scale Variant Image Pyramids (Gluckman 2006b), as well as Paris/Durand (Bae *et al.* 2006) [Note: right reference? read] and Zeev's newest work (Farbman *et al.* 2008). [Note: Finish off this section after reading the relevant papers.]

3.4.3 Wavelets

While pyramids are used extensively in computer vision applications, some people use *wavelet* decompositions as an alternative. Wavelets are filters that localize a signal in both space and frequency (like the Gabor filter in Table 3.2) and are defined over a hierarchy of scales. Wavelets provide a smooth way to decompose a signal into frequency components without blocking and are closely related to pyramids, as we will see shortly.

Wavelets were originally developed in the applied math and signal processing communities and were introduced to the computer vision community by Mallat (1989). Strang (1989), Rioul and Vetterli (1991), Meyer (1993) and Simoncelli and Adelson (1990b) all provide nice introductions to the subject along with historical reviews, while Chui (1992) provides a more comprehensive review and survey of applications. Sweldens (1997) provides a more recent description of the

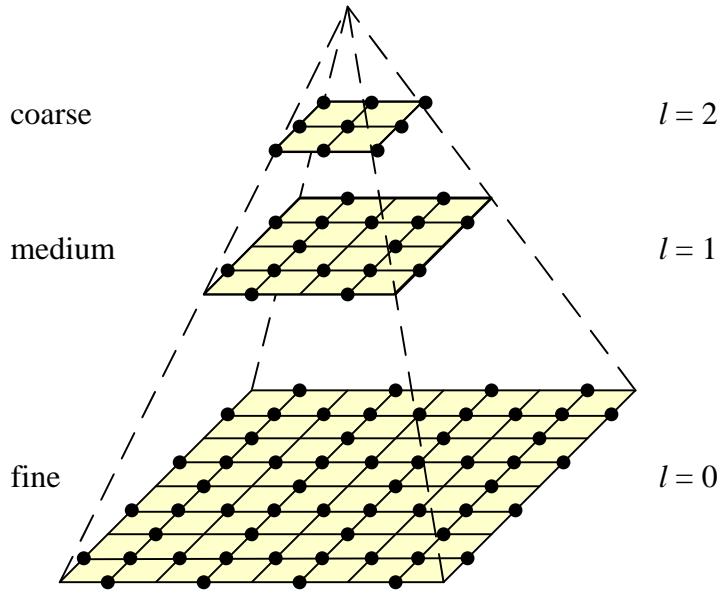


Figure 3.39: *Multiresolution wavelet pyramid. Each wavelet level stores $\frac{3}{4}$ of the original pixels (usually the horizontal, vertical, and mixed gradients), so that the total number of wavelet coefficients and original pixels is the same.*

lifting approach to wavelets I will describe shortly.

Wavelets are widely used in the computer graphics community to perform multi-resolution geometric processing (Stollnitz *et al.* 1996), and have also been used in computer vision for similar applications (Szeliski 1990b, Pentland 1994, Gortler and Cohen 1995, Yaou and Chang 1994, Lai and Vemuri 1997, Szeliski 2006b), as well as for multi-scale oriented filtering (Simoncelli *et al.* 1992) and de-noising (Portilla *et al.* 2003).

Since both image pyramids and wavelets decompose an image into multi-resolution descriptions that are localized in both space and frequency, how do they differ? The usual answer is that traditional pyramids are *overcomplete*, i.e., they use more pixels than the original image to represent the decomposition, whereas wavelets provide a *tight frame*, i.e., they keep the size of the decomposition the same as the image (Figures 3.39–3.41). However, some wavelet families are actually overcomplete in order to provide better shiftability or steering in orientation (Simoncelli *et al.* 1992). A better distinction, therefore, might be that wavelets are more orientation selective than regular band-pass pyramids.

How are two-dimensional wavelets constructed? Figure 3.40a shows a high-level diagram of one stage of the (recursive) coarse-to-fine construction (analysis) pipeline alongside the complementary re-construction (synthesis) stage. In this diagram, the high-pass filter followed by decimation keeps $\frac{3}{4}$ of the original pixels, while $\frac{1}{4}$ of the low-frequency coefficients are passed on to the

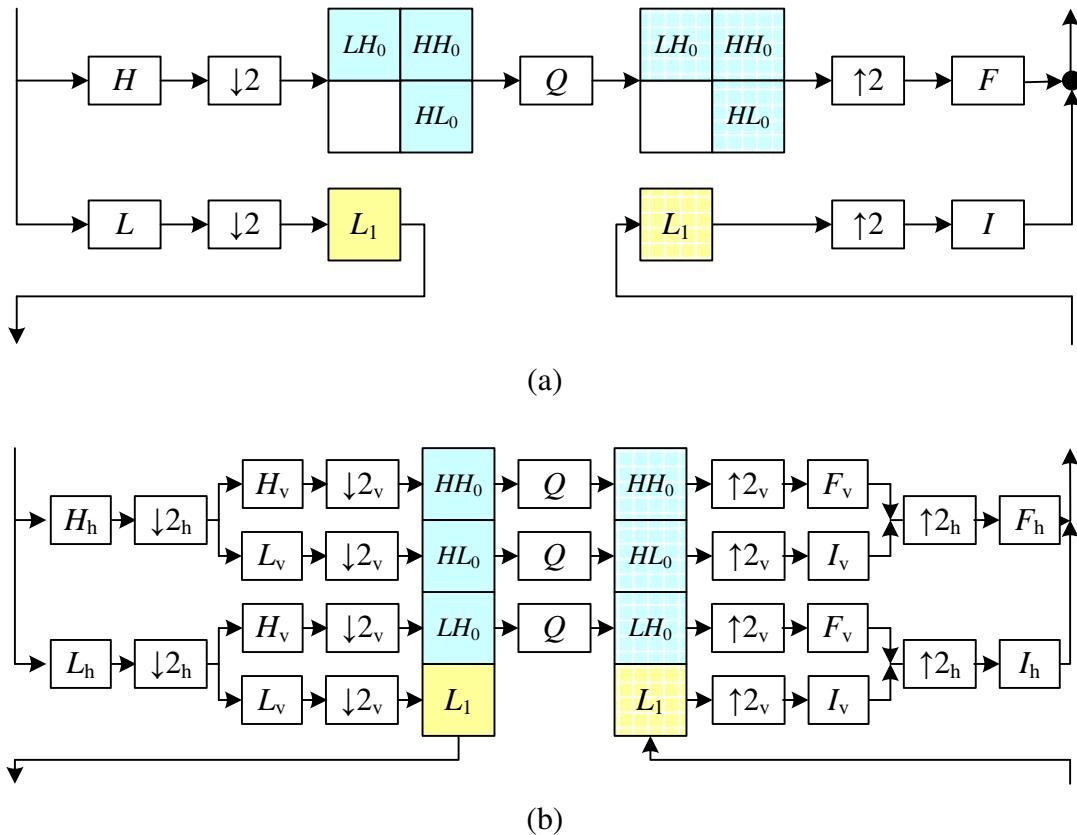


Figure 3.40: Two dimensional wavelet decomposition: (a) high-level diagram showing the low-pass and high-pass transforms as single boxes; (b) separable implementation, which involves first performing the wavelet transform horizontally and then vertically. The I and F boxes are the interpolation and filtering boxes required to re-synthesize the image from its wavelet components. [Note: Consider adding images where the colored boxes are.]

next stage for further analysis. In practice, the filtering is usually broken down into two separable sub-stages, as shown in Figure 3.40b. The resulting three wavelet images are sometimes called the high-high (HH), high-low (HL), and low-high (LH) images. Note how the high-low and low-high images accentuate the horizontal and vertical edges and gradients, while the high-high image contains the less frequently occurring mixed derivatives (Figure 3.41).

How are the high-pass H and low-pass L filters shown in Figure 3.40b chosen, and how can the corresponding reconstruction filters I and F be computed? Can such filters be designed that all have finite impulse responses? This topic has been the main subject of study in the wavelet community for over two decades; the answer depends largely on the intended application, e.g., whether the wavelets are being used for compression, image analysis (feature finding), or denoising. Simoncelli and Adelson (1990b), Table 4.1 show some good odd-length QMF (quadrature

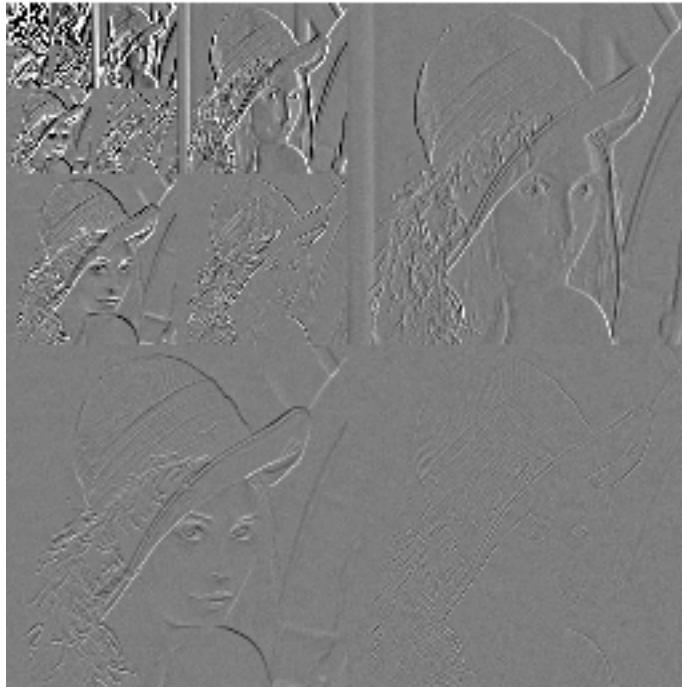


Figure 3.41: A wavelet decomposition of an image. The high-high components are in the lower right corner, while the base is in the upper left. Notice how the low-high and high-low components accentuate horizontal and vertical edges and gradients, while the high-high components store the less frequent mixed derivatives.

[Note: Replace this image with one of my own]

mirror filter) coefficients that seem to work well in practice.

Since the design of wavelet filters is such a tricky art, is there perhaps a better way? Indeed, a simpler procedure is to first split the signal into its even and odd components, and to then perform trivially reversible filtering operations on either sequence to produce what are called *lifted wavelets* (Figures 3.42–3.43). Sweldens (1996) gives a wonderfully understandable introduction to the subject (the *lifting scheme* for *second generation wavelets*), while (Sweldens 1997) contains a comprehensive review.

As Figure 3.42 demonstrates, rather than first filtering the whole input sequence (image) with high-pass and low-pass filters and then keeping the odd and even sub-sequences, the lifting scheme first splits the sequence into its even and odd sub-components. Filtering the even sequence with a low-pass filter L and subtracting the result from the even sequence is trivially reversible: simply perform the same filtering and then add the result back in. Furthermore, this operation can be performed in place, resulting in significant space savings. The same applies to filtering the even sequence with the correction filter C , which is used to ensure that the even sequence is low-

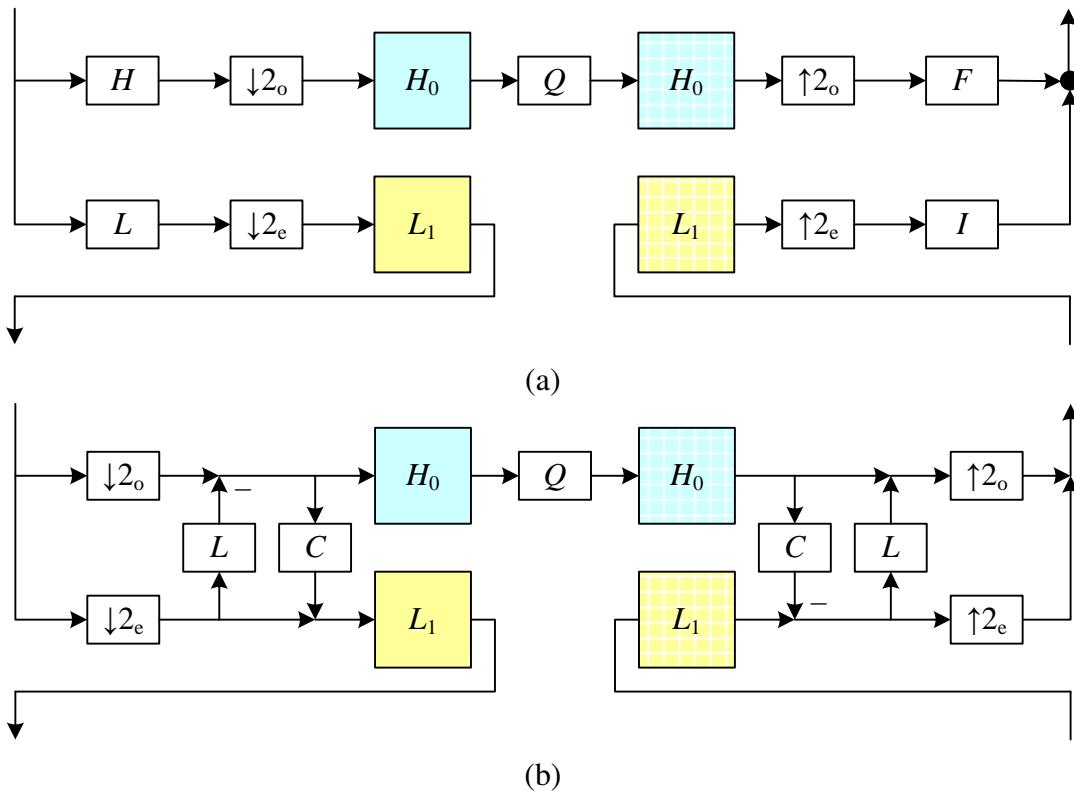


Figure 3.42: *One dimensional wavelet transform: (a) usual high-pass + low-pass filters followed by odd ($\downarrow 2_o$) and even ($\downarrow 2_e$) downsampling; (b) lifted version, which first selects the odd and even subsequences and then applies a low-pass prediction stage L and high-pass correction stage C in an easily reversible manner.*

pass. A series of such *lifting* steps can be used to create more complex filter responses with low computational cost and guaranteed reversibility.

This process can perhaps be more easily understood by considering the signal processing diagram in Figure 3.43. During analysis, the average of the even values is subtracted from the odd value to obtain a high-pass wavelet coefficient. However, the even samples still contain an aliased sample of the low-frequency signal. To compensate for this, a small amount of the high-pass wavelet is added back to the even sequence so that it is properly low-pass filtered. (It is easy to show that the effective low-pass filter is $[-\frac{1}{8}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, -\frac{1}{8}]$, which is indeed a low-pass filter.) During synthesis, the same operations are reversed with a judicious change in sign.

Of course, we need not restrict ourselves to just 2-tap filters. Figure 3.43 shows as light blue arrows additional filter coefficients that could optionally be added to the lifting scheme without affecting its reversibility. In fact, the low-pass and high-pass filtering operations can be interchanged, e.g., we could use a 5-tap cubic low-pass filter on the odd sequence (plus center value)

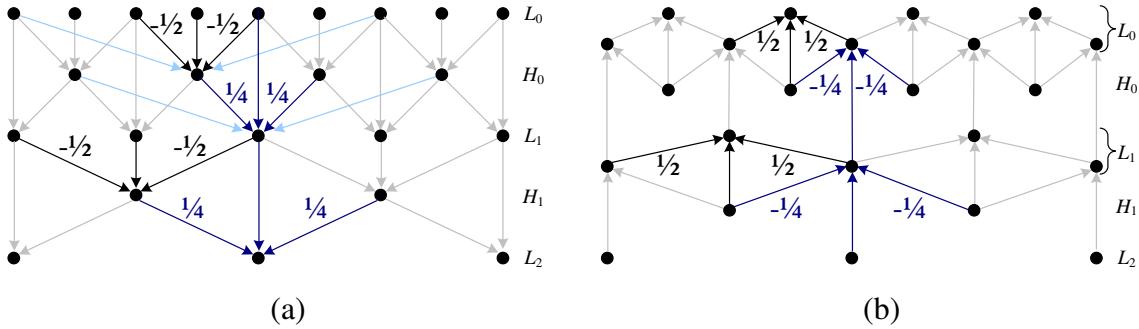


Figure 3.43: *Lifted transform shown as a signal processing diagram.* (a) *The analysis stage, which first predicts the odd value from its even neighbors and stores the difference wavelet, and then compensates the coarser even value by adding in a fraction of the wavelet.* (b) *The synthesis stage simply reverses the flow of computation and the signs of some of the filters and/or operations. The light blue lines show what happens if we use 4 taps for the prediction and correction instead of just 2.*

first, followed by a 4-tap cubic low-pass predictor to estimate the wavelet, although I haven't seen this scheme written down. [Note: Ask around: it may be well known.]

Lifted wavelets are called *second generation wavelets* because they can easily adapt to non-regular sampling topologies such as those that arise in computer graphics applications such as multi-resolution surface manipulation (Schröder and Sweldens 1995). It also turns out that lifted *weighted wavelets*, i.e., wavelets whose coefficients adapt to the underlying problem being solved, can be extremely effective at preconditioning the kinds of sparse linear systems that arise in optimization-based approaches to vision algorithms I discuss in §3.6 (Szeliski 2006b).

[Note: If needed, adjust previous figures, e.g., Figure 3.41 so that this page fills to near the bottom.]

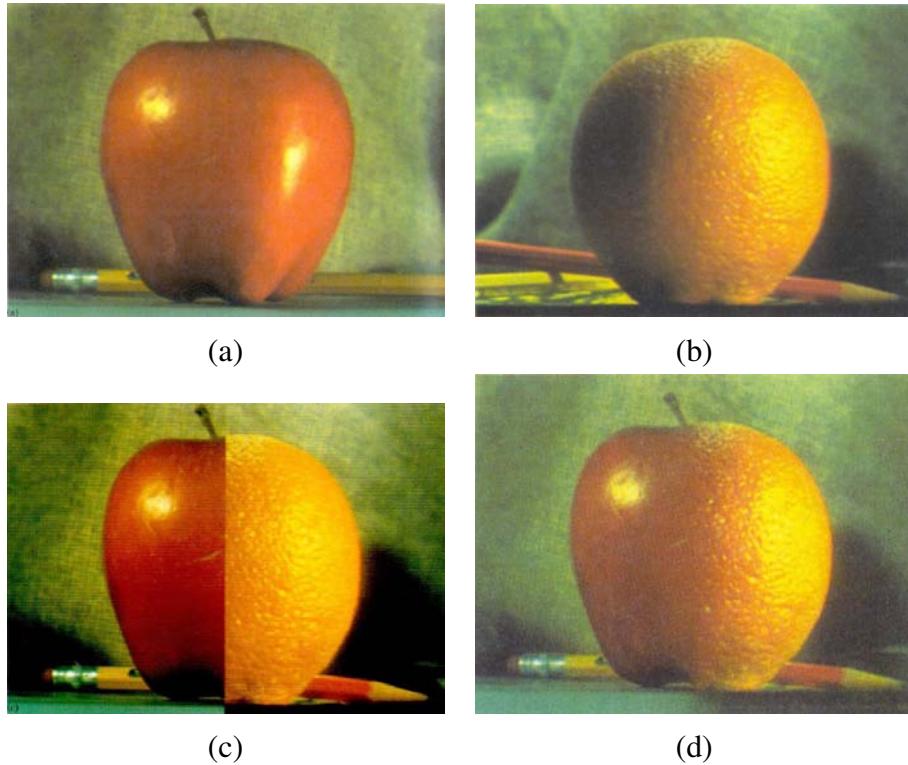


Figure 3.44: *Laplacian pyramid blending* (Burt and Adelson 1983b): (a–b) original images, (c) regular splice, (d) pyramid blend.

[Note: Get permission from ACM to publish these, find better quality originals.]

3.4.4 Application: Image blending

One of the most engaging and fun application of the Laplacian pyramid presented in §3.4.2 is the creation of blended composite images, as shown in Figure 3.44 (Burt and Adelson 1983b). While splicing the apple and orange image together along the midline produces a noticeable cut, *splining* them together (as Burt and Adelson (1983b) called their procedure) creates a beautiful illusion of a truly hybrid fruit. The key to their approach is that the low-frequency color variations between the red apple and the orange orange are smoothly blended, while the higher-frequency textures on each fruit are blended more quickly to avoid “ghosting” effects when two textures are overlaid.

To create the blended image, each source image is first decomposed into its own Laplacian pyramid (Figure 3.45, left and middle column). Each band is then multiplied by a smooth weighting function whose extent is proportional to the pyramid level. The simplest and most general way to create these weights is to take a binary mask image (Figure 3.46c) and to construct a *Gaussian* pyramid from this mask. Each Laplacian pyramid image is then multiplied by its corresponding Gaussian mask, and the sum of these two weighted pyramids is then used to construct the final

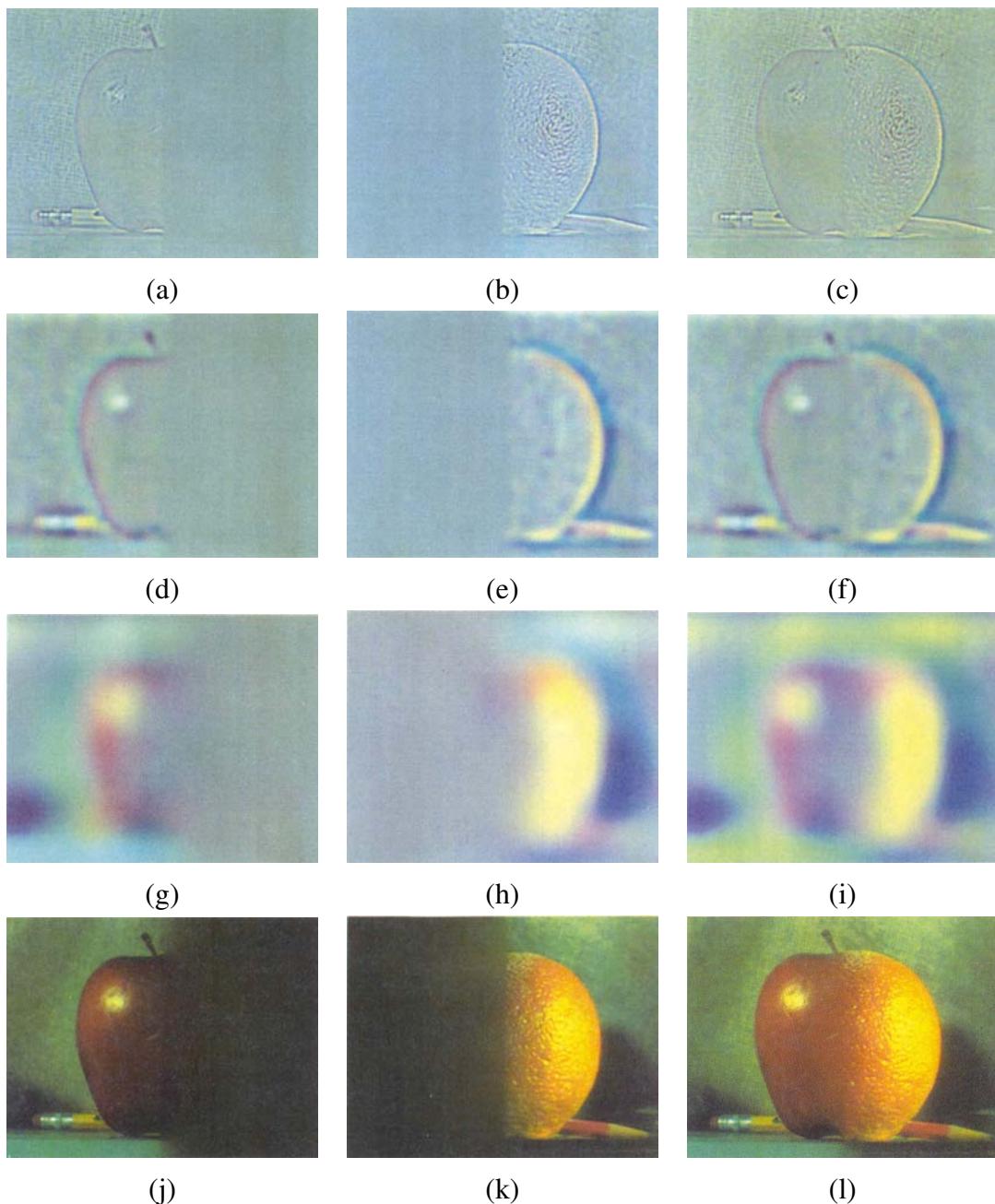


Figure 3.45: *Laplacian pyramid blending details (Burt and Adelson 1983b)*. The first three rows show the high, medium, and low frequency parts of the Laplacian pyramid (taken from levels 0, 2, and 4). The left and middle columns show the original apple and orange images weighted by the smooth interpolation functions, while the right column shows the averaged contributions.

[Note: Get permission from ACM to publish, find better quality originals. Some color images are also available in [burt-adelson-spline83.pdf](#)]

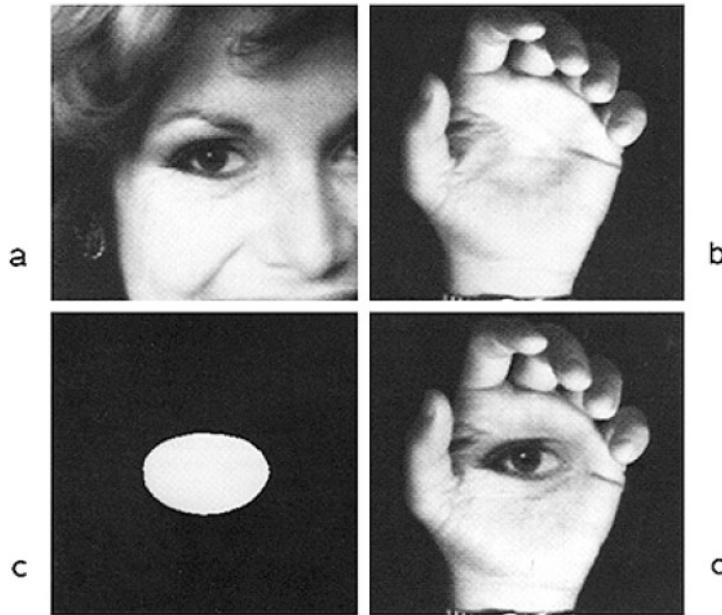


Fig. 8. The spline may be used to combine oddly shaped regions of very different images. The portion of Figure 8a within the region indicated by the mask in Figure 8c is inserted in the portion of Figure 8b which is outside this mask region (Figure 8d).

Figure 3.46: Laplacian pyramid blend of two images with an arbitrary shaped region (Burt and Adelson 1983b): (a–b) input images; (c)region mask; (d)blended image.

[Note: Get permission from ACM to publish these, find better quality originals.]

image (Figure 3.45, right column).

Figure 3.46 shows that this process can be applied to arbitrary mask images with surprising results. It is also straightforward to extend the pyramid blend to an arbitrary number of images whose pixel provenance is indicated by an integer-valued label image (see Exercise 3.20). This is particularly useful in image stitching and compositing applications where the exposures may vary between different images, as described in §8.3.3.

3.5 Geometric transformations

In the previous section, we saw how interpolation and decimation could be used to change the *resolution* of an image. In this section, we look at how to perform more general transformations such as image rotations or general warps. In contrast to the point processes we saw in §3.1, where the function applied to an image transforms the *range* of the image,

$$g(\mathbf{x}) = h(f(\mathbf{x})), \quad (3.86)$$

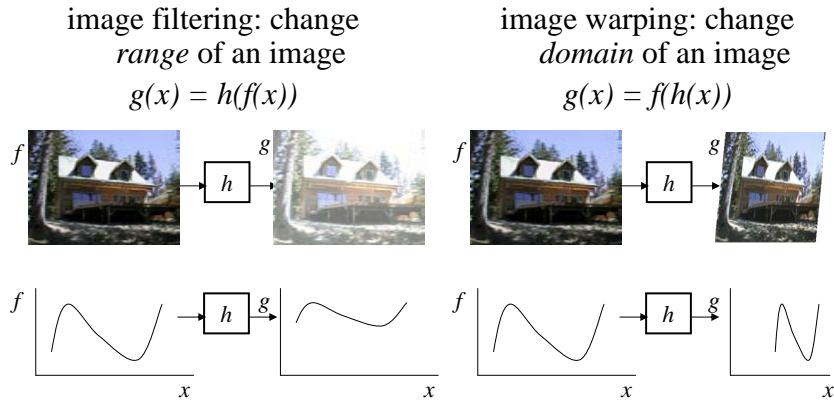


Figure 3.47: *Image warping involves modifying the domain of an image function rather than its range.*

here we look at functions that transform the *domain*,

$$g(\mathbf{x}) = f(\mathbf{h}(\mathbf{x})) \quad (3.87)$$

(see Figure 3.47).

We begin by studying the global *parametric* 2D transformation first introduced in §2.1.2. (Such transformations are called parametric because they are controlled by a small number of parameters.) We then turn our attention to more local general deformations such as those defined on meshes, §3.5.2. Finally, we show how image warps can be combined with cross-dissolves to create interesting *morphs* (in-between animations), §3.5.3. For readers interested in more details on these topics, there is an excellent survey by Heckbert (1986) as well as very accessible textbooks by Wolberg (1990), Gomes *et al.* (1999) and Akenine-Möller and Haines (2002). (Note that Heckbert’s survey is on *texture mapping*, which is what the topic of warping images onto surfaces is called in the computer graphics community.)

3.5.1 Parametric transformations

Parametric transformations apply a global deformation to an image, where the behavior of the transformation can be described with a small number of parameters. Figure 3.48 shows a sample of such transformations, which are based on the 2D geometric transformations shown in Figure 2.4. The formulas for these transformations were originally given in Table 2.1, and are reproduced here in Table 3.5 for ease of reference.

In general, given a transformation specified by a formula $\mathbf{x}' = \mathbf{h}(\mathbf{x})$ and a source image $f(\mathbf{x})$, how do we compute the values of the pixels in the new image $g(\mathbf{x})$, as given in (3.87)? Think about this for a minute before proceeding and see if you can figure it out.

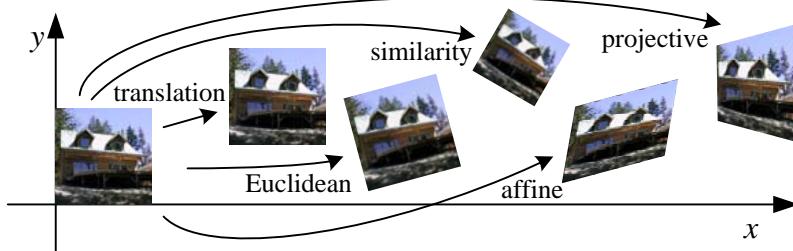


Figure 3.48: Basic set of 2D geometric image transformations

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$[I \mid t]_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$[R \mid t]_{2 \times 3}$	3	lengths + ...	
similarity	$[sR \mid t]_{2 \times 3}$	4	angles + ...	
affine	$[A]_{2 \times 3}$	6	parallelism + ...	
projective	$[\tilde{H}]_{3 \times 3}$	8	straight lines	

Table 3.5: Hierarchy of 2D coordinate transformations. The 2×3 matrices are extended with a third $[0^T \ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

If you are like most people, you will come up with an algorithm that looks something like this:

For every pixel x in $f(x)$,

1. compute the destination location $x' = h(x)$;
2. copy the pixel $f(x)$ to $g(x')$.

This process is called *forward warping* or *forward mapping* and is shown in Figure 3.49a. Can you think of any problems with this approach?

In fact, this approach suffers from several limitations. The process of copying a pixel $f(x)$ to a location x' in g is not well defined when x' has a non-integer value. What do we do in such a case? What would you do?

You can round the value of x' to the nearest integer coordinate and copy the pixel there, but the resulting image has severe aliasing and pixels that jump around a lot when animating the transformation. You can also “distribute” the value among its four nearest neighbors in a weighted (bilinear) fashion, keeping track of the per-pixel weights and normalizing at the end. This technique is called *splatting*, and is sometimes used for volume rendering in the graphics community

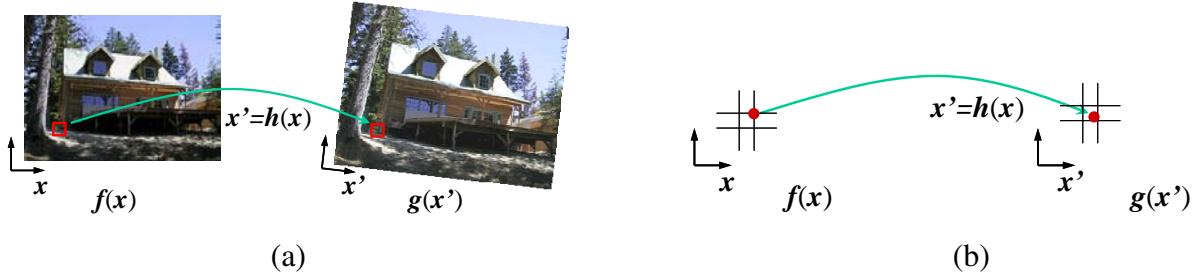


Figure 3.49: *Forward warping algorithm*: (a) a pixel $f(\mathbf{x})$ is copied to its corresponding location $\mathbf{x}' = \mathbf{h}(\mathbf{x})$ in image $g(\mathbf{x}')$; (b) detail of the source and destination pixel locations.

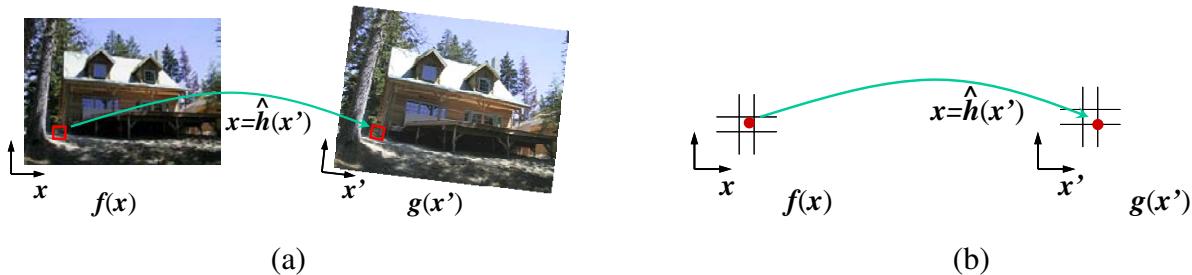


Figure 3.50: *Inverse warping algorithm*: (a) a pixel $g(\mathbf{x}')$ is sampled from its corresponding location $\mathbf{x} = \hat{\mathbf{h}}(\mathbf{x}')$ in image $f(\mathbf{x})$; (b) detail of the source and destination pixel locations.

(Rusinkiewicz and Levoy 2000). [Note: read the paper, and add the reference to Levoy's original splatting paper] Unfortunately, it suffers from both moderate amounts of aliasing and a fair amount of blur (loss of high-resolution detail).

The second major problem with forward warping is the appearance of cracks and holes, especially when magnifying an image. Filling such holes with their nearby neighbors can lead to further aliasing and blurring.

What can we do instead? A preferable solution is to use *inverse warping* (Figure 3.50), where each pixel in the destination image $g(\mathbf{x}')$ is sampled from the original image $f(\mathbf{x})$, i.e.,

For every pixel \mathbf{x}' in $g(\mathbf{x}')$,

1. compute the source location $\mathbf{x} = \hat{\mathbf{h}}(\mathbf{x}');$
2. resample $f(\mathbf{x})$ at location \mathbf{x} and copy to $g(\mathbf{x}')$.

How does this differ from the forward warping algorithm? For one thing, since $\hat{\mathbf{h}}(\mathbf{x}')$ is (presumably) defined for all pixels in $g(\mathbf{x}')$, we no longer have holes. More importantly, resampling an image at non-integer locations is a well studied problem (general image interpolation §3.4.1), and high-quality filters that control aliasing can be used.

Where does the function $\hat{\mathbf{h}}(\mathbf{x}')$ come from? Quite often, it can simply be computed as the inverse of $\mathbf{h}(\mathbf{x})$. In fact, all of the parametric transforms listed in Table 3.5 have closed form

solutions for the inverse transform: simply take the inverse of the 3×3 matrix specifying the transform.

In other cases, it is preferable to formulate the problem of image warping as that of resampling a source image $f(\mathbf{x})$ given a mapping $\mathbf{x} = \hat{\mathbf{h}}(\mathbf{x}')$ from destination pixels \mathbf{x}' to source pixels \mathbf{x} . For example, in optical flow (§7.4), we estimate the flow field as the location of the *source* pixel which produced the current pixel whose flow is being estimated, as opposed to computing the *destination* pixel where it is going to. Similarly, when correcting for radial distortion (§2.1.5), we calibrate the lens by computing for each pixel in the final (undistorted) image the corresponding pixel location in the original (distorted) image.

What kinds of interpolation filters are suitable for the resampling process? Any of the filters we studied in §3.4.1 can be used, including nearest neighbor, bilinear, bicubic, and windowed sinc functions. While bilinear is often used for speed (e.g., inside the inner loop of a patch tracking algorithm, §7.1.3), bicubic and windowed sinc are preferable where visual quality is important.

To compute the value of $f(\mathbf{x})$ at a non-integer location \mathbf{x} , we simply apply our usual FIR resampling filter,

$$g(x, y) = \sum_{k,l} f(k, l)h(x - k, y - l), \quad (3.88)$$

where (x, y) are the sub-pixel coordinate values and $h(x, y)$ is some interpolating or smoothing kernel. Recall from §3.4.1 that when decimation is being performed, the smoothing kernel is stretched and re-scaled according to the downsampling rate r . [Note: A figure here that shows the different behaviors of interpolation and decimation (one kernel stays fixed, the other is rate adaptive) might be helpful here.]

Unfortunately, for a general (non-zoom) image transformation, the resampling rate r is not well defined. Consider a transformation that stretches the x dimensions while squashing the y . The resampling kernel should be performing regular interpolation along the x dimension while performing smoothing (to anti-alias the blurred image) in the y direction. This gets even more complicated for the case of general affine or perspective transforms.

What can we do? Fortunately, Fourier analysis can help. The two-dimensional generalization of the one-dimensional *domain scaling* law given in Table 3.1 is

$$g(\mathbf{A}\mathbf{x}) \Leftrightarrow |\mathbf{A}|^{-1}G(\mathbf{A}^{-T}\mathbf{f}). \quad (3.89)$$

For all of the transform in Table 3.5 except for perspective, the matrix \mathbf{A} is already defined. For perspective transformations, the matrix \mathbf{A} is the linearized *derivative* of the perspective transformation (Figure 3.51a), i.e., the local affine approximation to the stretching induced by the projection (Heckbert 1986, Wolberg 1990, Gomes *et al.* 1999, Akenine-Möller and Haines 2002).

To prevent aliasing, we need to pre-filter the image $f(\mathbf{x})$ with a filter whose frequency response is the projection of the final desired spectrum through the \mathbf{A}^{-T} transform (Szeliski *et al.* 2008b). In

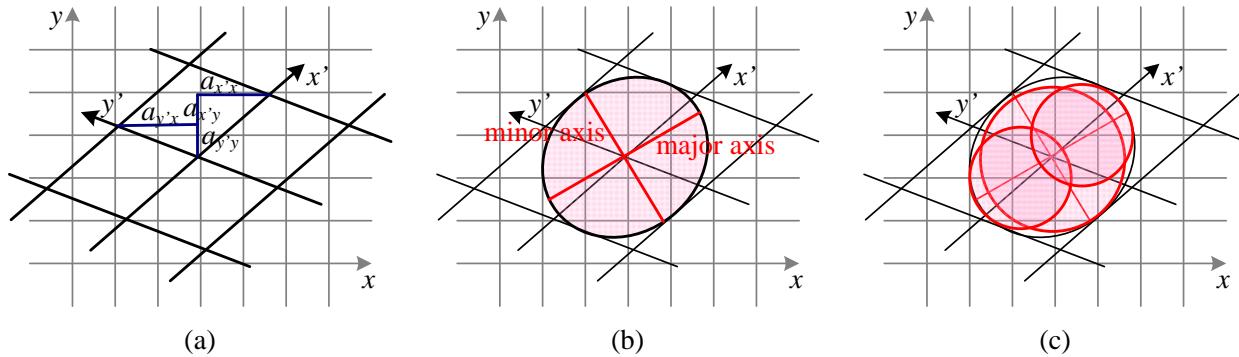


Figure 3.51: *Anisotropic texture filtering*: (a) Jacobian of transform \mathbf{A} and the induced horizontal and vertical resampling rates $\{a_{x'x}, a_{x'y}, a_{y'x}, a_{y'y}\}$; (b) elliptical footprint of an EWA smoothing kernel; (c) anisotropic filtering using multiple samples along the major axis. Image pixels lie at line intersections.

general (for non-zoom transforms), this filter is non-separable and hence is very slow to compute. Therefore, a number of approximations to this filter are used in practice, include MIP-mapping, elliptically weighted Gaussian averaging, and anisotropic filtering (Akenine-Möller and Haines 2002).

MIP-mapping

MIP-mapping was first proposed by Williams (1983) as a means to rapidly pre-filter images being used for *texture mapping* in computer graphics. A MIP-map¹³ is a standard image pyramid (Figure 3.34), where each level is pre-filtered with a high-quality filter (rather than a poorer quality approximation such as Burt and Adelson's 5-tap binomial). To resample an image from a MIP-map, a scalar estimate of the resampling rate r is first computed. For example, r can be the maximum of the absolute values in \mathbf{A} (which will suppress aliasing), or it can be the minimum (which will reduce blurring). Akenine-Möller and Haines (2002) discuss these issues in more detail.

Once a resampling rate has been specified, a *fractional* pyramid level is computed using the base 2 logarithm,

$$l = \log_2 r. \quad (3.90)$$

One simple solution is to then resample the texture from the next higher or lower pyramid level, depending on whether it is preferable to reduce aliasing or blur. A better solution is to resample *both* images, and to then linearly blend them using the fractional component of l . Since most MIP-map implementation use bilinear resampling within each level, this approach is usually called

¹³ The MIP stands for *multi in parvo*, many-in-one.

trilinear MIP-mapping. Computer graphics rendering APIs such as OpenGL and Direct3D have parameters that can be used to select which variant of MIP-mapping (and of the sampling rate r computation) should be used, depending on the desired speed vs. quality tradeoff. Exercise 3.22 has you discuss some of these tradeoffs in more detail.

Elliptical Weighted Average

The Elliptical Weighted Average (EWA) filter invented by [Greene and Heckbert \(1986\)](#) is based on the observation that the affine mapping $\mathbf{x} = \mathbf{A}\mathbf{x}'$ defines a skewed two-dimensional coordinate system in the vicinity of each source pixel \mathbf{x} (Figure 3.51a). For every destination pixel \mathbf{x}' , the ellipsoidal projection of a small pixel grid in \mathbf{x}' onto \mathbf{x} is computed (Figure 3.51b). This is then used to filter the source image $g(\mathbf{x})$ with a Gaussian whose inverse covariance matrix is this ellipsoid.

Despite its reputation as a high-quality filter ([Akenine-Möller and Haines 2002](#)), we have found in our work ([Szeliski et al. 2008b](#)) that because a Gaussian kernel is used, the technique suffers simultaneously from both blurring and aliasing, compared to higher-quality filters. The EWA is also quite slow, although faster variants based on MIP-mapping have been proposed (see ([Szeliski et al. 2008b](#)) for some additional references). [Note: decide if I want to include a figure from ([Szeliski et al. 2008b](#)), or perhaps something else (bikes?) from the submission material. Also, give a pointer to the supplementary book materials.]

Anisotropic filtering

An alternative approach to filtering oriented textures, which is sometimes implemented in graphics hardware (GPUs), different x and y sampling rates, potentially on a tilted grid) is to use anisotropic filtering ([Barkans 1997](#), [Akenine-Möller and Haines 2002](#)). In this approach, several samples at different resolutions (fractional levels in the MIP-map) are combined along the major axis of the EWA Gaussian (Figure 3.51c).

Multi-pass transforms

The optimal approach to warping images without excessive blurring or aliasing is to adaptively pre-filter the source image at each pixel using an ideal low-pass filter, i.e., an oriented skewed sinc or low-order (e.g., cubic) approximation (Figure 3.51a). Figure 3.52, taken from ([Szeliski et al. 2008b](#)), based on related figures in ([Heckbert 1989](#), [Dodgson 1992](#)), shows how this works in one dimension. The signal is first (theoretically) interpolated to a continuous waveform, then (ideally) low-pass filtered to below the new Nyquist rate, and then re-sampled to the final desired resolution. In practice, the interpolation and decimation steps are concatenated into a single *polyphase* digital filtering operation. ([Szeliski et al. 2008b](#)).

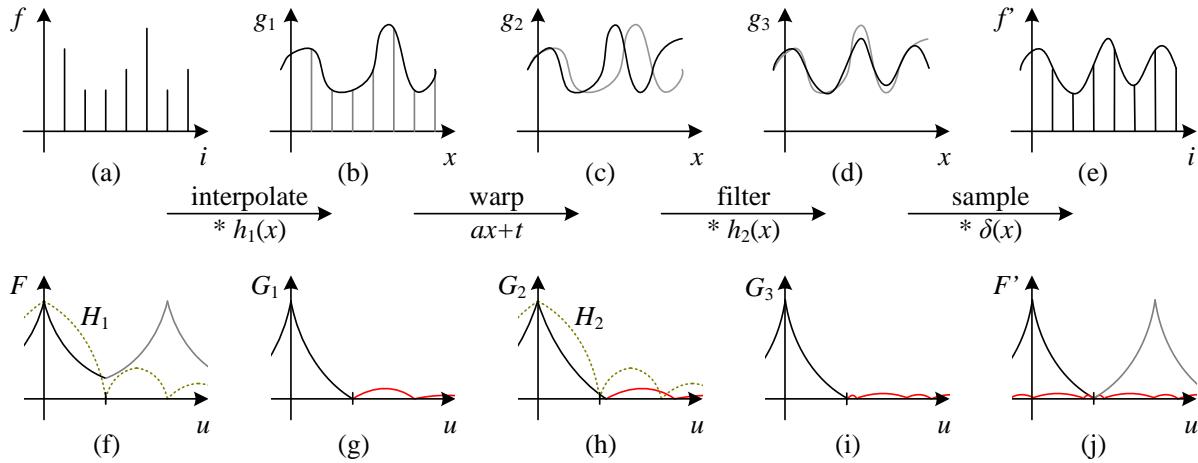


Figure 3.52: *One-dimensional signal resampling* (Szeliski et al. 2008b): (a) original sampled signal $f(i)$; (b) interpolated signal $g_1(x)$; (c) warped signal $g_2(x)$; (d) filtered signal $g_3(x)$; (e) sampled signal $f'(i)$. The corresponding spectra are shown below the signals in figures (f–j), with the aliased portions shown in red.

For parametric transforms, the oriented two-dimensional filtering and resampling operations can be approximated using a series of one-dimensional resampling and shearing transforms (Catmull and Smith 1980, Heckbert 1989, Wolberg 1990, Gomes et al. 1999, Szeliski et al. 2008b). The advantage of using a series of 1-D transforms is that these are much more efficient (in terms of basic arithmetic operations) than large non-separable two-dimensional filter kernels.

In order to prevent aliasing, however, it may be necessary to upsample in the opposite direction before applying a shearing transformation (Szeliski et al. 2008b). Figure 3.53 shows this process for a rotation, where a vertical upsampling stage is added before the horizontal shearing (and upsampling) stage. The upper image shows the appearance of the letter being rotated, while the lower image shows its corresponding Fourier transform.

[Note: show animation (GIFs or PNGs ?) of original and rotated by $\pm 2^\circ$; aliasing will be visible as loss of sharpness and as “crawling”. See supplemental materials for (Szeliski et al. 2008b): probably not needed in the book.]

3.5.2 Mesh-based warping

While parametric transforms specified by a small number of global parameters have many uses, local deformations with more degrees of freedom are often required.

Consider, for example, changing the appearance of a face from a frown to a smile (Figure 3.54a). What's needed in this case is to curve the corners of the mouth upwards while leaving

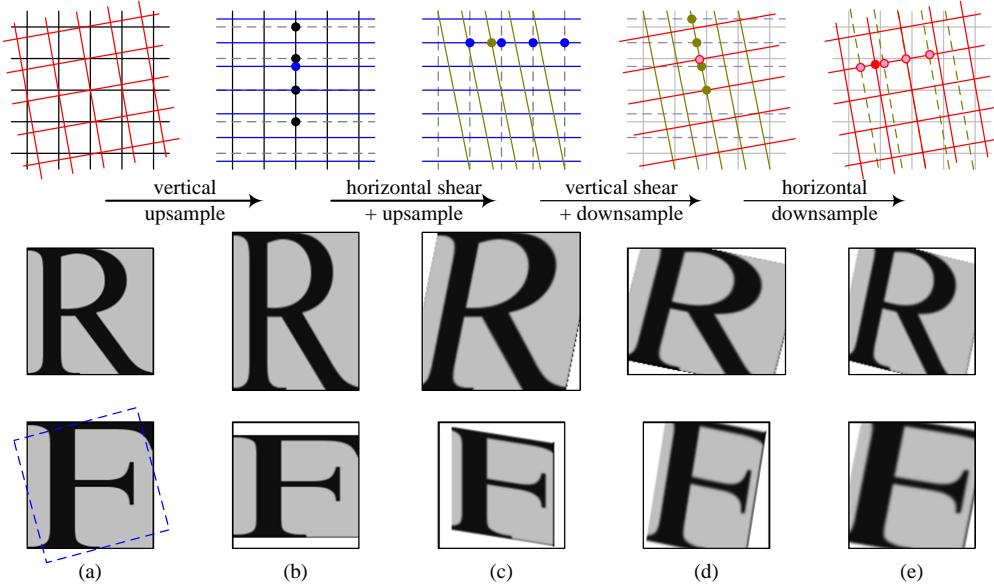


Figure 3.53: 4-pass rotation (Szeliski et al. 2008b): (a) original pixel grid, image, and its Fourier transform; (b) vertical upsampling; (c) horizontal shear and upsampling; (d) vertical shear and downsampling; (e) horizontal downsampling. The general affine case looks similar except that the first two stages perform general resampling.

the rest of the face intact. (See (Rowland and Perrett 1995, Pighin *et al.* 1998, Blanz and Vetter August 1999, Leyvand *et al.* 2008) for some more sophisticated examples of changing facial expression and appearance.) To perform such a transformation, different amounts of motion are required in different parts of the image. Figure 3.54 shows some of the commonly used approaches.

The first approach, shown in Figure 3.54a–b, is to specify a *sparse* set of corresponding points in both images. The displacement of these points can then be interpolated to a dense *displacement field*, §7, using a variety of techniques (Nielson 1993). One possibility is to *triangulate* the set of points in one image (de Berg *et al.* 2006, Litwinowicz and Williams 1994, Buck *et al.* 2000) and to use an *affine* motion model (Table 3.5), specified by the three triangle vertices, inside each triangle. If the destination image is triangulated according to the new vertex locations, an inverse warping algorithm (Figure 3.50) can be used. If the source image is triangulated and as used as a *texture map*, computer graphics rendering algorithms can be used to draw the new image (but care must be taken along triangle edges to avoid potential aliasing).

Alternative methods for interpolating a sparse set of displacements include moving nearby quadrilateral mesh vertices, as shown in Figure 3.54a, using *variational* (energy minimizing) interpolants such as regularization (Litwinowicz and Williams 1994) §3.6.1, or using locally weighted (*radial basis function*) combinations of displacements (Nielson 1993). (See §11.5.1 for additional

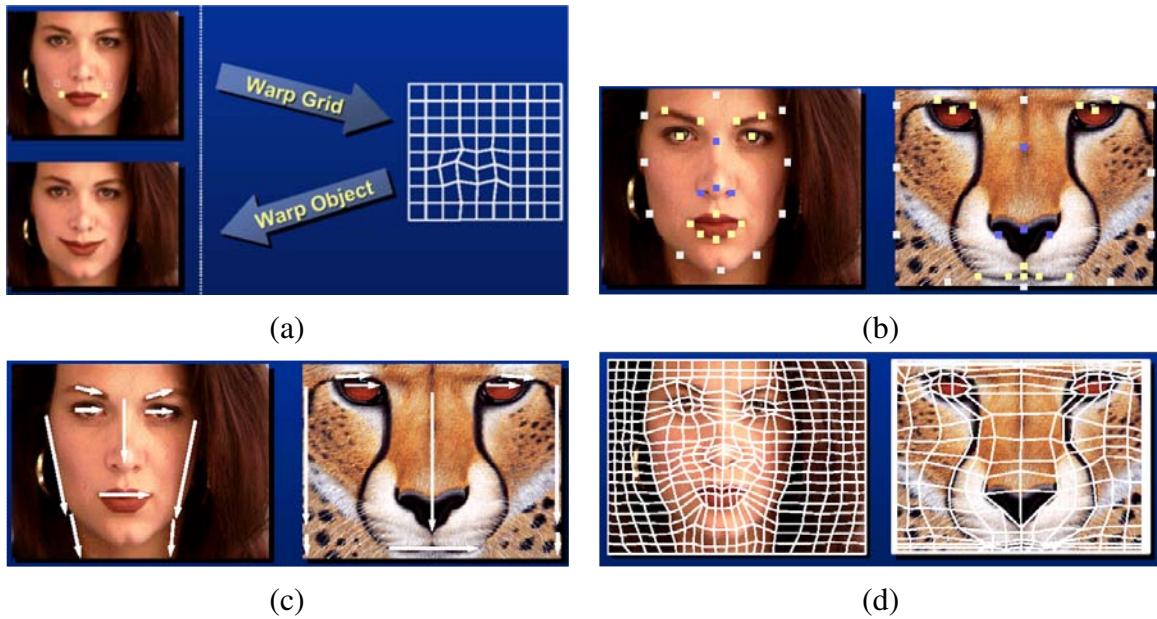


Figure 3.54: *Image warping alternatives:* (a) sparse control points \rightarrow deformation grid; (b) denser set of control point correspondences; (c) oriented line correspondences (Beier and Neely 1992); (d) uniform quadrilateral grid.

[Note: Versions of these figures appear in (Gomes et al. 1999). I'm not sure where I got these particular figures; looks like they are from a blue background PPT, perhaps from one of their courses.]

scattered data interpolation techniques). If quadrilateral meshes are used, it may be desirable to interpolate displacements down to individual pixel values using a smooth interpolant such as a quadratic B-spline (Farin 1992).¹⁴

In some cases, e.g., if a dense depth map has been estimated for an image (Shade et al. 1998), we only know the forward displacement for each pixel. As mentioned before, drawing source pixels at their destination location, i.e., forward warping (Figure 3.49), suffers from several potential problems, including aliasing and the appearance of small cracks. An alternative technique in this case is to forward warp the *displacement field* (or depth map) to its new location, fill small holes in the resulting map, and then use inverse warping to perform the resampling (Shade et al. 1998). The reason that this generally works better than forward warping is that displacement fields tend to be much smoother than images, so the aliasing introduced during the forward warping of the displacement field is much less noticeable.

A second approach to specifying displacements for local deformations is to use corresponding

¹⁴ Note that the *block-based* motion models used by many video compression standards (Le Gall 1991) can be thought of as a 0th order (piecewise constant) displacement field.

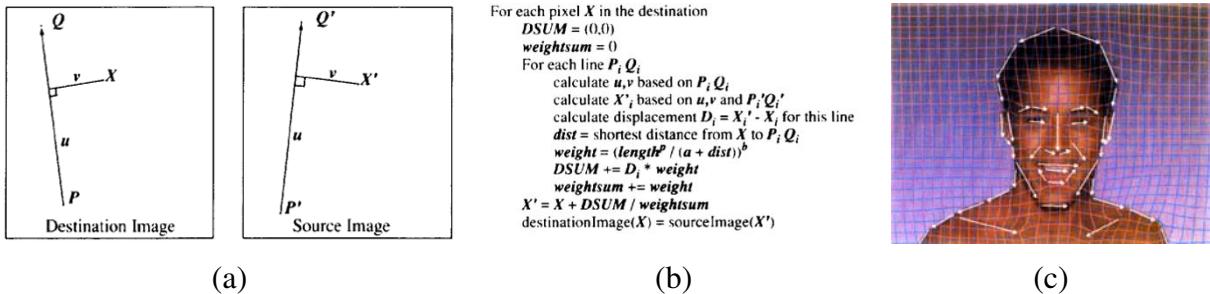


Figure 3.55: Line-based image warping (Beier and Neely 1992): (a) distance computation and position transfer; (b) rendering algorithm; (c) two intermediate warps used for morphing.

oriented line segments, as shown in Figures 3.54c and 3.55. Pixels along each line segment are transferred from source to destination exactly as specified, and other pixels are warped using a smooth interpolation of these displacements. Each line segment correspondence specifies a translation, rotation, and scaling, i.e., a *similarity transform* (Table 3.5), for pixels in its vicinity, as shown in Figure 3.55a. Line segments influence the overall displacement of the image using a weighting function that depends on the minimum distance to the line segment (v in Figure 3.55a if $u \in [0, 1]$, else the shorter of the two distances to P and Q).

For each pixel X , its target location X' for each line correspondence is computed along with a weight that depends on the distance and the line segment length (Figure 3.55b). The weighted average of all target locations X'_i then becomes the final destination location. Note that while Beier and Neely describe this algorithm as a forward warp, an equivalent algorithm can be written by sequencing through the destination pixels. (The resulting warps will not be identical because line lengths and/or distances to lines may be different.) Exercise 3.23 has you implement the Beier-Neely (line-based) warp and compare it to a number of other local deformation methods.

One final possibility for specifying displacement fields is to use a mesh specifically *adapted* to the underlying image content, as shown in Figure 3.54d. Specifying such meshes by hand can involve a fair amount of work; Gomes *et al.* (1999) describe an interactive system for doing this. Once the two meshes have been specified, intermediate warps can be generated using linear interpolation, and the displacements at mesh nodes can be interpolated using splines.

3.5.3 Application: Feature-based morphing

While warps can be used to change the appearance or to animate a *single* image, even more powerful effects can be obtained by warping and blending two or more images using a process now commonly known as *morphing* (Beier and Neely 1992, Gomes *et al.* 1999).

Figure 3.56 shows the essence of image morphing. Instead of simply cross-dissolving between two images, which leads to ghosting as shown in the top row, each image is warped toward the

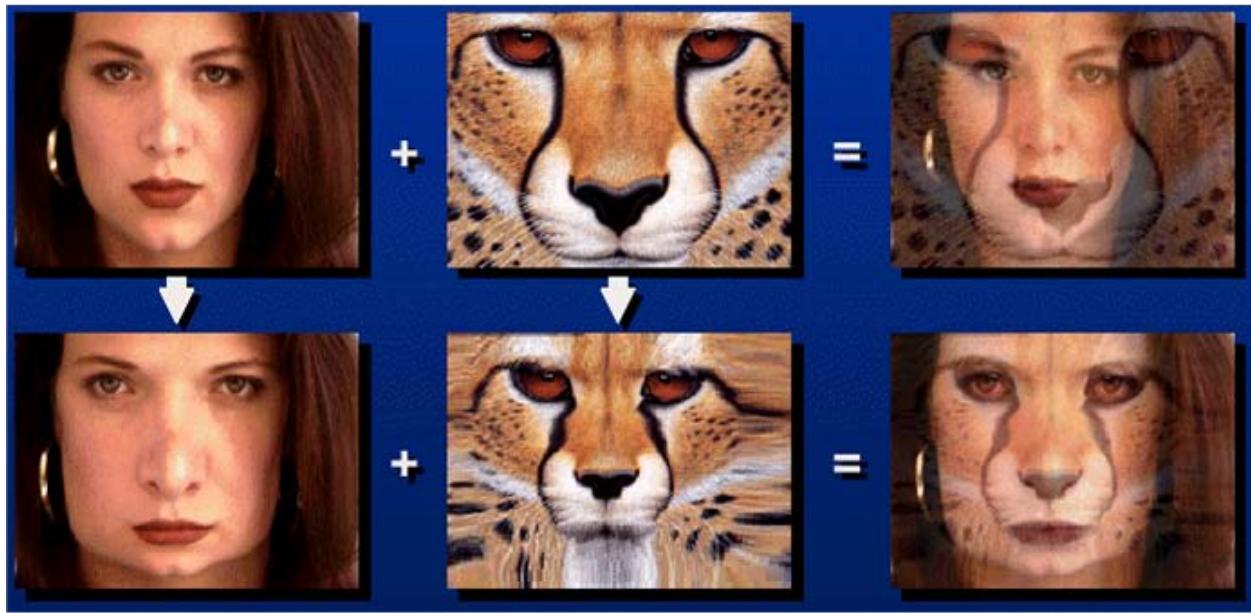


Figure 3.56: *Image morphing* (Gomes et al. 1999). Top row: if the two images are just blended, visible ghosting results. Bottom row: both images are first warped to the same intermediate location (e.g., halfway towards the other image) and the resulting warped images are then blended resulting in a seamless morph.

other image before blending, as shown in the bottom row. If the correspondences have been set up well (using any of the techniques shown in Figure 3.54), corresponding features are always aligned, and so no ghosting results (Beier and Neely 1992).

The above process is repeated for each intermediate frame being generated during a morph, using different blends (and amounts of deformation) at each interval. Let $t \in [0, 1]$ be the time parameter that describes the sequence of interpolated frames. The weighting functions for the two warped images in the blend go as $(1 - t)$ and t . Conversely, the amount of motion that image 0 undergoes at time t is t of the total amount of motion that is specified by the correspondences. However, some care must be taken in defining what it means to partially warp an image towards a destination, especially if the desired motion is far from linear (Sederberg et al. 1993). Exercise 3.25 has you implement a morphing algorithm and test it out under such challenging conditions.

3.6 Global optimization

So far in this chapter, I have covered a large number of image processing operators that take as input one or more images and produce some filtered or transformed version of these images. In many applications, it is more useful to first *formulate* the goals of the desired transformation using

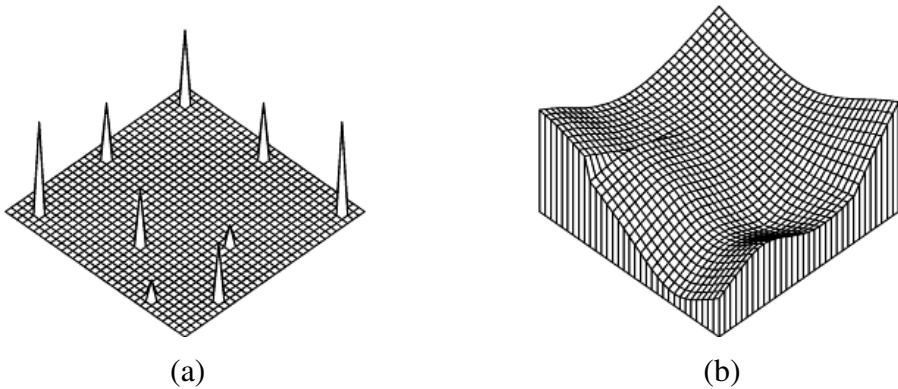


Figure 3.57: A simple surface interpolation problem: (a) nine data points of various height scattered on the grid; (b) second order controlled-continuity thin-plate spline interpolator, with a tear along its left edge and a crease along its right (Szeliski 1989).

some optimization criterion, and to then find or infer the solution that best meets this criterion.

In this final section, I present two different (but closely related) variants on this idea. The first, which is often called *regularization* (§3.6.1), constructs a continuous global energy function that describes the desired characteristics of the solution and then finds a minimum energy solution using sparse linear systems or related iterative techniques. The second formulates the problem using Bayesian statistics, modeling both the noisy measurement process that produced the input images as well as *prior assumptions* about the solution space, which are often encoded using a *Markov random field* (§3.6.2).

Examples of such problems include surface interpolation from scattered data (Figure 3.57), image denoising and the restoration of missing regions (Figures 3.60–3.61), as well as the segmentation of images into foreground and background regions (Figure 3.64).

3.6.1 Regularization

The theory of regularization was first developed by statisticians trying to fit models to data that severely underconstrained the solution space (Tikhonov and Arsenin 1977). Consider, for example, finding a smooth surface that passes through (or near) a set of measured data points (Figure 3.57). Such a problem is called *ill-posed*, since many possible surfaces can fit this data. Since small changes in the input can sometimes lead to large changes in the fit (e.g., if we use polynomial interpolation), such problems are also often *ill-conditioned*. Finally, since we are trying to recover the unknown function $f(x, y)$ from which the data point $d(x_i, y_i)$ were sampled, such problems are also often called *inverse problems*. (Many computer vision task can be viewed as inverse problems, since we are trying to recover a full description of the 3D world from a limited set of images.)

In order to quantify what it means to find a *smooth* solution, we can define a norm on the solution space. For one dimensional functions $f(x)$, we can integrate the squared first derivative of the function,

$$\mathcal{E}_1 = \int f_x^2(x) dx \quad (3.91)$$

or perhaps integrate the squared second derivative,

$$\mathcal{E}_2 = \int f_{xx}^2(x) dx. \quad (3.92)$$

(Here, I use subscripts to denote differentiation.) Such energy measures are examples of *functionals*, which are operators that map functions to scalar values. They are also often called *variational formulations*, because they measure the variation (non-smoothness) in a function.

In two dimensions (e.g., for images, flow fields, or surfaces), the corresponding smoothness functionals are

$$\mathcal{E}_1 = \int f_x^2(x, y) + f_y^2(x, y) dx dy = \int \|\nabla f(x, y)\|^2 dx dy \quad (3.93)$$

and

$$\mathcal{E}_2 = \int f_{xx}^2(x, y) + 2f_{xy}^2(x, y) + f_{yy}^2(x, y) dx dy, \quad (3.94)$$

where the mixed $2f_{xy}^2$ term is needed to make the measure rotationally invariant.

The first derivative norm is often called the *membrane*, since interpolating a set of data points using this measure results in a tent-like structure. (In fact, this formula is a small-deflection approximation to the surface area, which is what soap bubbles minimize.) The second order norm is called the *thin-plate spline*, since it approximates the behavior of thin plates (e.g., flexible steel) under small deformations. A blend of the two is called the *thin plate spline under tension*, and versions of these formulas where each derivative term is multiplied by a local weighting function are called *controlled-continuity splines* (Terzopoulos 1988). [Note: Check whether I should be citing a different paper by Demetri.] Figure 3.57 shows a simple example of a controlled-continuity interpolator fit to nine scattered data points. In practice, it is more common to find first order smoothness terms used with images and flow fields (§7.4), and second order smoothness associated with surfaces (§11.5.1).

In addition to the smoothness term, regularization also requires a data term (or *data penalty*). For scattered data interpolation (Nielson 1993), the data term measures the distance between the function $f(x, y)$ and a set of data points $d_i = d(x_i, y_i)$,

$$\mathcal{E}_d = \sum_i [f(x_i, y_i) - d_i]^2. \quad (3.95)$$

For a problem like noise removal, a continuous version of this measure can be used,

$$\mathcal{E}_d = \int [f(x, y) - d(x, y)]^2 dx dy. \quad (3.96)$$

To obtain a global energy that can be minimized, the two energy terms are usually added together,

$$\mathcal{E} = \mathcal{E}_d + \lambda \mathcal{E}_s, \quad (3.97)$$

where \mathcal{E}_s is the *smoothness penalty* (either \mathcal{E}_1 or \mathcal{E}_2 or some weighted blend), and λ is called the *regularization parameter*, which controls how smooth the solution should be.

In order to find the minimum of this continuous problem, the function $f(x, y)$ is usually first discretized on a regular grid.¹⁵ The most principled way to perform this discretization is to use *finite element analysis*, i.e., to approximate the function with a piecewise continuous spline, and to then perform the analytic integration (Bathe 2007).

Fortunately, for both the first order and second order smoothness functionals, the judicious selection of appropriate finite elements results in particularly simple discrete forms (Terzopoulos 1983). The corresponding *discrete* smoothness energy functions become

$$\begin{aligned} E_1 &= \sum_{i,j} s_x(i, j)[f(i+1, j) - f(i, j) - g_x(i, j)]^2 \\ &\quad + s_y(i, j)[f(i, j+1) - f(i, j) - g_y(i, j)]^2 \end{aligned} \quad (3.98)$$

and

$$\begin{aligned} E_2 &= \sum_{i,j} c_x(i, j)[f(i+1, j) - 2f(i, j) + f(i-1, j)]^2 \\ &\quad + 2c_m(i, j)[f(i+1, j+1) - f(i+1, j) - f(i, j+1) + f(i, j)]^2 \\ &\quad + c_y(i, j)[f(i, j+1) - 2f(i, j) + f(i, j-1)]^2. \end{aligned} \quad (3.99)$$

The optional smoothness weights $s_x(i, j)$ and $s_y(i, j)$ control the location of horizontal and vertical tears (or weaknesses) in the surface. For other problems, such as colorization (Levin *et al.* 2004) and interactive tone mapping (Lischinski *et al.* 2006a), they control the smoothness in the interpolated chroma or exposure field, and are often set inversely proportional to the local luminance gradient strength. For second order problems, the crease variables $c_x(i, j)$, $c_m(i, j)$, and $c_y(i, j)$ control the locations of creases in the surface (Terzopoulos 1988, Szeliski 1990a).

The data values $g_x(i, j)$ and $g_y(i, j)$ are gradient data terms (constraints) used by algorithms such as photometric stereo §11.1.1, HDR tone mapping §9.2.1 (Fattal *et al.* 2002), Poisson blending §8.3.3 (Pérez *et al.* 2003), and gradient-domain blending §8.3.3 (Levin *et al.* 2004). They are set to zero when just discretizing the conventional first order smoothness functional (3.93).

The two dimensional discrete data energy is written as

$$E_d = \sum_{i,j} w(i, j)[f(i, j) - d(i, j)]^2, \quad (3.100)$$

¹⁵ The alternative of using *kernel basis functions* centered on the data points (Boult and Kender 1986, Nielson 1993) is discussed in more detail in §11.5.1.

where the local weights $w(i, j)$ control how strongly the data constraint is enforced. These values are set to zero where there is no data, and can be set to the inverse variance of the data measurements when there is (see (Szeliski 1989) and §3.6.2).

The total energy of the discretized problem can now be written as a *quadratic form*

$$E = E_d + \lambda E_s = \mathbf{x}^T \mathbf{A} \mathbf{x} - 2\mathbf{x}^T \mathbf{b} + c, \quad (3.101)$$

where $\mathbf{x} = [f(0, 0) \dots f(m-1, n-1)]$ is called the *state vector*.¹⁶

The sparse symmetric positive-definite matrix \mathbf{A} is called the *Hessian* since it encodes the second derivative of the energy function.¹⁷ For the one-dimensional first order problem, \mathbf{A} is tridiagonal, while for the two-dimensional first order problem, it is multi-banded with 5 non-zero entries per row. We call \mathbf{b} the *weighted data vector*. Minimizing the above quadratic form is equivalent to solving the sparse linear system

$$\mathbf{A} \mathbf{x} = \mathbf{b}, \quad (3.102)$$

which can be done using a variety of sparse matrix techniques, such as multi-grid (Briggs 1987) and hierarchical preconditioners (Szeliski 2006b), as described in Appendix A.5.

While regularization was first introduced to the vision community by Terzopoulos (1986) for the problem of surface interpolation, it was quickly adopted by other vision researchers for such varied problems as edge detection §4.2, optical flow §7.4, and shape from shading §11.1 (Poggio *et al.* 1985, Horn and Brooks 1986, Terzopoulos 1986, Brox *et al.* 2004). Poggio *et al.* (1985) also showed how the discrete energy defined by (3.99) and (3.100) could be implemented in a resistive grid, as shown in Figure 3.58. In computational photography §9, regularization and its variants are commonly used to solve problems such as high-dynamic range tone mapping (Fattal *et al.* 2002, Lischinski *et al.* 2006a), Poisson and gradient-domain blending (Pérez *et al.* 2003, Levin *et al.* 2004, Agarwala *et al.* 2004), colorization (Levin *et al.* 2004), and natural image matting (Levin *et al.* 2008).

Robust regularization

While regularization is most commonly formulated using quadratic (L_2) norms, c.f. the squared derivatives in (3.91–3.94) and squared differences in (3.99–3.100), it can also be formulated using non-quadratic *robust* penalty functions (Appendix B.3). For example, (3.99) can be generalized to

$$E_{1r} = \sum_{i,j} s_x(i, j) \rho(f(i+1, j) - f(i, j)) \quad (3.103)$$

¹⁶ We use \mathbf{x} instead of f because this is the more common form in the numerical analysis literature (Golub and Van Loan 1996).

¹⁷ In numerical analysis, \mathbf{A} is called the *coefficient* matrix (Saad 2003), while in finite element analysis (Bathe 2007), it is called the *stiffness* matrix.

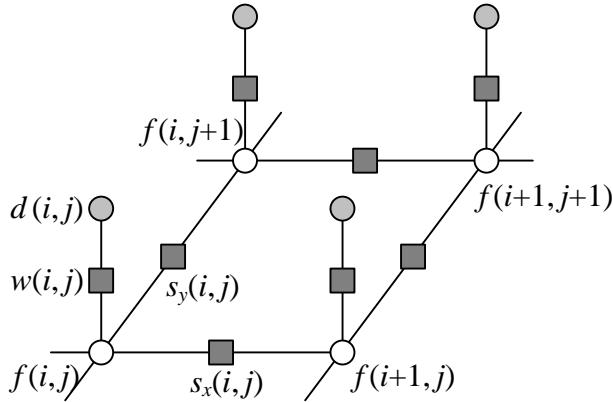


Figure 3.58: *Graphical model interpretation of first order regularization.* The white circles are the unknowns $f(i, j)$ while the dark circles are the input data $d(i, j)$. In the resistive grid interpretation, the d and f values encode input and output voltages, and the black squares denote resistors whose conductance is set to $s_x(i, j)$, $s_y(i, j)$, and $w(i, j)$. In the spring-mass system analogy, the circles denote elevations and the black squares denote springs. The same graphical model can also be used to depict a first-order Markov random field (Figure 3.59).

$$+ s_y(i, j)\rho(f(i, j + 1) - f(i, j)),$$

where $\rho(x)$ is some monotonically increasing penalty function. For example, the family of norms $\rho(x) = |x|^p$ are called p -norms. When $p < 2$, the resulting smoothness terms become more piecewise continuous rather than totally smooth, which can better model the discontinuous nature of images, flow fields, and 3D surfaces.

An early example of robust regularization is the *graduated non-convexity* (GNC) algorithm introduced by [Blake and Zisserman \(1987\)](#). Here, the norms on the data and derivatives are clamped to a maximum value

$$\rho(x) = \min(x^2, V). \quad (3.104)$$

Because the resulting problem is highly non-convex (it has many local minima), a *continuation* methods is proposed, where a quadratic norm (which is convex) is gradually replaced by the non-convex robust norm. (Around the same time, [Terzopoulos \(1988\)](#) was also using continuation to infer the tear and crease variables in his surface interpolation problems.)

Today, it is more common to use the L_1 ($p = 1$) norm, which is often called *total variation* ([Chan et al. 2001](#)). [Note: Need to read up more about these techniques. [Chan et al. \(2001\)](#) use a $\sqrt{\nabla^2 I + a^2}$ norm defined on a discrete lattice, i.e. where ∇^2 is actually implemented as $\sum_{j \in \mathcal{I}_i} (I_j - I_i)^2$. Look at Tschumperlé's articles ([Tschumperlé and Deriche 2005](#), [Tschumperlé 2006](#)), and look in CV-bib for “total variation”. ([Kafifay et al. 2007](#)) cites other work, such as [Buades \(Buades et al. 2005, Buades et al. 2008\)](#).] Other norms, whose influence (derivative)

more quickly decays to zero are presented in (Black and Rangarajan 1996, Black *et al.* 1998) and discussed in (Appendix B.3).

[Note: Either here or in Appendix B.3, plot a bunch of different penalty functions, along with their influence functions—see (Black and Rangarajan 1996, Black *et al.* 1998).]

Even more recently, so called *hyper-Laplacian* norms with $p < 1$ have been proposed, based on the observation that the log-likelihood distribution of image derivatives follows a $p < 1$ slope (and is hence a hyper-Laplacian distribution) (Levin *et al.* 2004, Weiss and Freeman 2007). [Note: Which of Anat's papers talk about this? Probably easiest to look at her home page.] Such norms have an even stronger tendency to prefer large discontinuities over small ones. [Note: Issues about discretization: applying p -norms separately to each finite difference is not the same.]

[Note: Is it worth showing a simple 1-D example with different norms applied to the same step edge?]

While least-squares regularized problems using L_2 norms can be solved using linear systems, other p -norms require different iterative techniques such as iteratively reweighted least squares (IRLS) or Levenberg-Marquardt. Such techniques are discussed in §5.1.3 and Appendix A.3.

3.6.2 Markov Random Fields

As we have just seen, regularization, which involves the minimization of energy functionals defined over (piecewise) continuous functions, can be used to formulate and solve a variety of low-level computer vision problems. An alternative technique is to formulate a *Bayesian* model, which separately models the noisy image formation (*measurement*) process, as well as assumes a statistical *prior* model over the solution space. In particular, in this section we look at priors based on *Markov random fields*, whose log-likelihood can be described using local neighborhood interaction (or penalty) terms (Kindermann and Snell 1980, Geman and Geman 1984, Li 1995, Szeliski *et al.* 2008c).

The use of Bayesian modeling has several potential advantages over regularization (see also Appendix B). The ability to model measurement processes statistically enables us to extract the maximum information possible from each measurement, rather than just guessing what weighting to give the data. Similarly, the parameters of the prior distribution can often be *learned* by observing samples from the class we are modeling (Tappen 2007). Furthermore, because our model is probabilistic, it is possible to estimate (in principle) complete probability *distributions* over the unknown being recovered, and in particular to model the *uncertainty* in the solution, which can be useful in latter processing stages. Finally, Markov random field models can be defined over *discrete* variables such as image labels (where the variables have no proper ordering), for which regularization does not apply.

Recall from (3.67) in §3.3.1 (or see Appendix B.4), that according to Bayes' Rule, the *posterior* distribution for a given set of measurement \mathbf{y} , $p(\mathbf{y}|\mathbf{x})$, combined with a prior $p(\mathbf{x})$ over the unknowns \mathbf{x} , is given by

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}, \quad (3.105)$$

where $p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$ is a normalizing constant used to make the $p(\mathbf{x}|\mathbf{y})$ distribution *proper* (integrate to 1). Taking the negative logarithm of both sides of (3.105), we get

$$-\log p(\mathbf{x}|\mathbf{y}) = -\log p(\mathbf{y}|\mathbf{x}) - \log p(\mathbf{x}) + C, \quad (3.106)$$

which is the *negative posterior log likelihood*.

To find the most likely (*maximum a posteriori* or MAP) solution \mathbf{x} given some measurements \mathbf{y} , we simply minimize this negative log likelihood, which can also be thought of as an *energy*,

$$E(\mathbf{x}, \mathbf{y}) = E_d(\mathbf{x}, \mathbf{y}) + E_p(\mathbf{x}). \quad (3.107)$$

(We drop the constant C because its value does not matter during energy minimization.) The first term $E_d(\mathbf{x}, \mathbf{y})$ is the *data energy* or *data penalty*, and measures the negative log likelihood that the data were observed given the unknown state \mathbf{x} . The second term $E_p(\mathbf{x})$ is the *prior energy*, and plays a role analogous to the smoothness energy in regularization. Note that the MAP estimate may not always be desirable, since it selects the “peak” in the posterior distribution rather than some more stable statistic—see the discussion in Appendix B.4. [Note: Does (Bishop 2006) also discuss this?]

For image processing applications, the unknowns \mathbf{x} are the set of output pixels

$$\mathbf{x} = [f(0, 0) \dots f(m-1, n-1)],$$

and the data are (in the simplest case) the input pixels

$$\mathbf{y} = [d(0, 0) \dots d(m-1, n-1)]$$

(see Figure 3.59).

For a Markov random field, the probability $p(\mathbf{x})$ is a *Gibbs* or *Boltzmann distribution*, whose negative log likelihood (according to the Hammersley-Clifford Theorem) can be written as a sum of pairwise interaction potentials,

$$E_p(\mathbf{x}) = \sum_{(i,j)} \sum_{(k,l) \in \mathcal{N}(i,j)} V_{i,j,k,l}(f(i,j), f(k,l)), \quad (3.108)$$

where $\mathcal{N}(i, j)$ denotes the *neighbors* of pixel (i, j) . (In fact, the general version of the Theorem says that the energy may have to be evaluated over a larger set of *cliques*, which depend on the *order*

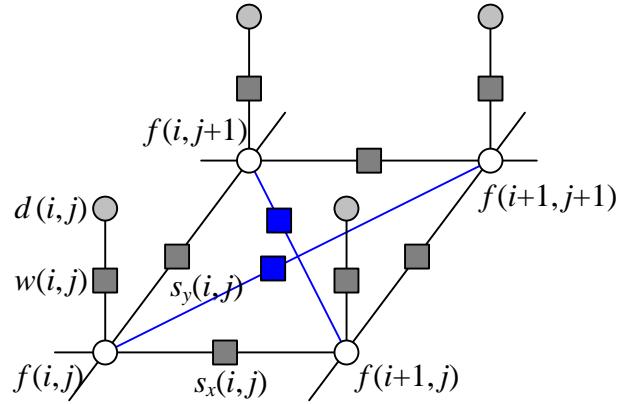


Figure 3.59: *Graphical model for a \mathcal{N}_4 neighborhood Markov random field. (The blue edges are added for an \mathcal{N}_8 neighborhood.) The white circles are the unknowns $f(i,j)$, while the dark circles are the input data $d(i,j)$. The $s_x(i,j)$ and $s_y(i,j)$ black boxes denote arbitrary interaction potentials between adjacent nodes in the random field, and the $w(i,j)$ denote the data penalty functions. The same graphical model can also be used to depict a discrete version of a first-order regularization problem (Figure 3.58).*

of the Markov Random field (Kindermann and Snell 1980, Geman and Geman 1984, Bishop 2006). However, we will not study higher-order cliques in this book.)

The most commonly used neighborhood in Markov random field modeling is the \mathcal{N}_4 neighborhood, where each pixel in the field $f(i,j)$ interacts only with its immediate neighbors. Figure 3.59, which we previously used in Figure 3.58 to illustrate the discrete version of first-order regularization, shown an \mathcal{N}_4 MRF. The $s_x(i,j)$ and $s_y(i,j)$ black boxes denote arbitrary *interaction potentials* between adjacent nodes in the random field, and $w(i,j)$ denotes the data penalty function. These square nodes can also be interpreted as *factors* in a *factor graph* version of the (undirected) graphical model (Bishop 2006), which is another name for interaction potentials. (Strictly speaking, the factors are (im-proper) probability functions whose product is the (un-normalized) posterior distribution.)

While \mathcal{N}_4 neighborhoods are most commonly used, in some applications, \mathcal{N}_8 (or even higher order) neighborhoods perform better at tasks such as image segmentation because they can better model discontinuities at different orientations (Boykov and Kolmogorov 2003).

[Note: Do I need to say more about equivalence of energy and Bayesian methods? I think it's implicit in what I've presented.]

Binary MRFs

The simplest possible example of a Markov random field is a binary field. Examples of such fields include 1-bit (black and white) scanned document images as well as images segmented into foreground and background.

To denoise a scanned image, we set the data penalty to reflect the agreement between the scanned and final images,

$$E_d(i, j) = w\delta(f(i, j), d(i, j)) \quad (3.109)$$

and the smoothness penalty to reflect the agreement between neighboring pixels

$$E_p(i, j) = E_x(i, j) + E_y(i, j) = s\delta(f(i, j), f(i + 1, j)) + s\delta(f(i, j), f(i, j + 1)). \quad (3.110)$$

Figure 3.60 (inspired by (Bishop 2006, Figure 8.30)) shows an example of an original (noise-free) image along with its noisy version $d(i, j)$, where 10% of the pixels have been randomly flipped.

How do we set the parameters in the data and smoothness terms? One possibility is to let the energies be the negative log-likelihoods of their respective probabilities, i.e., we set $w = -\log 0.1$ and $s = -\log ???$, where the fraction ??? was obtained by counting the percentage of *edges* in Figure 3.60a. (This is not *really* cheating. In practice, we would use a *training* corpus of image to set these parameters §9.1.1.)

[Note: Replace the ??? above once I've generated the real figures.]

Once we have formulated the energy, how do we minimize it? The simplest approach is to perform gradient descent, flipping one state at a time if it produces a lower energy. This approach is known as *iterated conditional modes* (ICM) (Kittler and Föglein 1984, Besag 1986), or *highest confidence first* (HCF) (Chou and Brown 1990) if the pixel with the largest energy decrease is selected first. *[Note: read (Kittler and Föglein 1984) to make sure it's appropriate; it's cited in (Bishop 2006). Also, re-read (Chou and Brown 1990) to see if it's the highest energy drop or really the most confident pixel that's selected.]*

Unfortunately, these downhill methods tend to get easily stuck in local minima. An alternative approach is to add some randomness to the process, which is known as *stochastic gradient descent* (Metropolis *et al.* 1953, Geman and Geman 1984). When the amount of noise is decreased over time, this technique is known as *simulated annealing* (Kirkpatrick *et al.* 1983, Carnevali *et al.* 1985, Wolberg and Pavlidis 1985, Swendsen and Wang 1987) and was first popularized in computer vision by Geman and Geman (1984) and later applied to stereo matching by Barnard (1989), among others.

Even this technique, however, does not perform that well (Boykov *et al.* 2001). For binary images, a much better technique, introduced to the computer vision community by Boykov *et al.* (2001) is to re-formulate the energy minimization as a *max-flow/min-cut* graph optimization problem (Greig *et al.* 1989). (This technique has informally come to be known as *graph cuts* in the

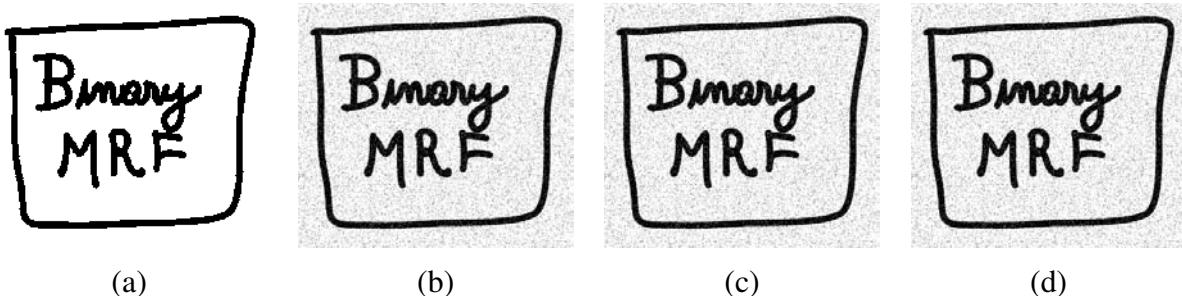


Figure 3.60: *Binary image denoising*: (a) original image; (b) noise corrupted image; (c) de-noised using ICM; (d) de-noised using graph cuts.

[Note: Still need to generate the real noisy and de-noised images.]

computer vision community.) For simple energy functions (e.g., those where the energy is identical for neighboring unlike pixels), this algorithm is guaranteed to produce the *global minimum*. Kolmogorov and Zabih (2004) formally characterize the class of binary energy potentials (*regularity conditions*) for which these results hold, while newer work by Komodakis *et al.* (2007) and Rother *et al.* (2007) provide good algorithms for the cases when they do not. Appendix B.6.4 gives more details about how to convert the energy function corresponding to a binary MRF into a graph problem that can be solved by max-flow/min-cut algorithms.

Figures 3.60c and 3.60d show the results of running ICM and graph cuts on our sample binary image denoising problem. Another binary labeling problem, namely that of segmenting an image into foreground and background regions given region color statistics (Boykov and Jolly 2001) is discussed in §4.5.4.

In addition to the above mentioned techniques, a number of other optimization approaches have been developed for MRF energy minimization, such as (loopy) belief propagation and dynamic programming (for one-dimensional problems). These are discussed in more detail in Appendix B.6 as well as the recent comparative survey paper by Szeliski *et al.* (2008c).

Ordinal-valued MRFs

In addition to binary images, Markov Random Fields can be applied to ordinal-valued labels such as grayscale images or depth maps. (The term ordinal implies that the labels have an implied ordering, e.g., higher values are lighter pixels. In the next subsection, I look at unordered labels such as source image labels for image compositing.)

In many cases, it is common to extend the binary data and smoothness prior terms as

$$E_d(i, j) = w(i, j)\rho_d(f(i, j) - d(i, j)) \quad (3.111)$$

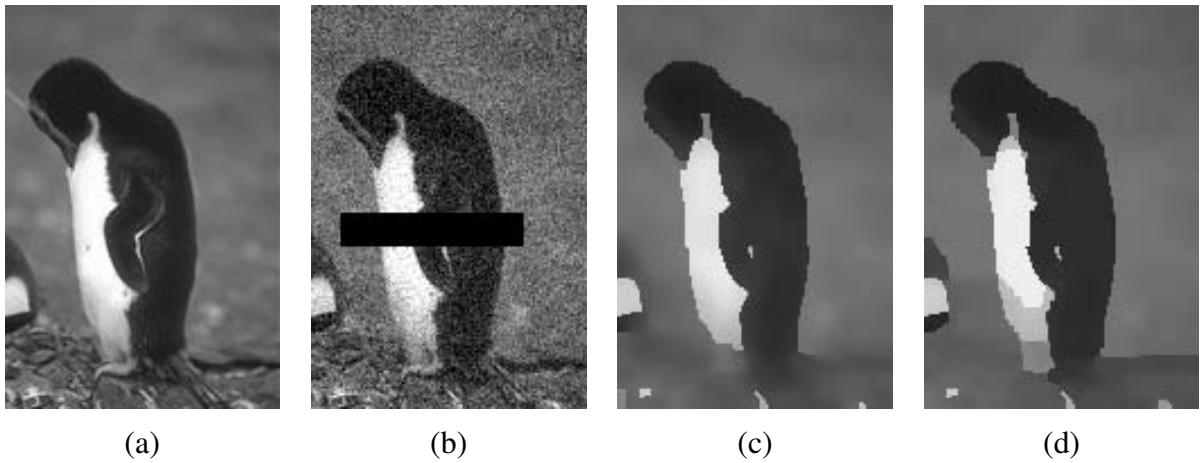


Figure 3.61: *Grayscale image denoising*: (a) original image; (b) noise corrupted image with missing data (black bar); (c) restored using loopy belief propagation; (d) restored using expansion move graph cuts. Images are from <http://vision.middlebury.edu/MRF/results/> (Szeliski et al. 2008c).

[Note: Add the image for gradient descent as well?]

and

$$E_p(i, j) = s_x(i, j)\rho_p(f(i, j) - f(i + 1, j)) + s_y(i, j)\rho_p(f(i, j) - f(i, j + 1)), \quad (3.112)$$

which are robust generalizations of the quadratic penalty terms (3.100) and (3.99) first introduced in §3.6.1, (3.104). As before, the $w(i, j)$, $s_x(i, j)$ and $s_y(i, j)$ weights can be used to locally control the data weighting and the horizontal and vertical smoothness. Instead of using a quadratic penalty, however, a general monotonically increasing penalty function $\rho()$ is used. (Different functions can be used for the data and smoothness terms.) For example, ρ_p can be a hyper-Laplacian penalty

$$\rho_p(d) = |d|^p, \quad p < 1, \quad (3.113)$$

which better encodes the distribution of gradients (mainly edges) in an image than either a quadratic or linear (total variation) penalty.¹⁸ Levin *et al.* (2004) use such a penalty to separate a transmitted and reflected image (Figure 7.8) by encouraging gradients to lie in one or the other image, but not both. More recently, Levin *et al.* (2007) use the hyper-Laplacian as a prior for image deconvolution (deblurring). For the data penalty, ρ_d can be quadratic (to model Gaussian noise), or the log of a contaminated Gaussian (Appendix B.3).

¹⁸ Note that unlike a quadratic penalty, the sum of the horizontal and vertical derivative p -norms is not rotationally invariant. A better approach may be to locally estimate the gradient direction and to impose different norms on the perpendicular and parallel components (Roth and Black 2007), which makes this a Conditional Random Field.

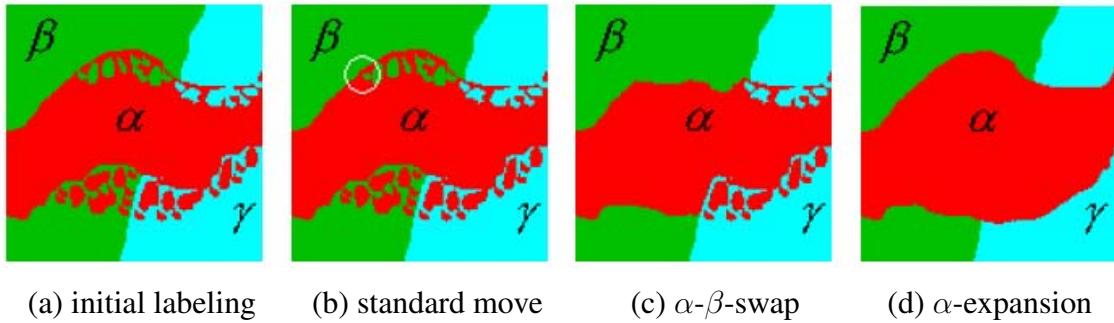


Figure 3.62: *Multi-level graph optimization from (Boykov et al. 2001)*: (a) *initial problem configuration*; (b) *the standard move only changes one pixel*; (c) *the α - β -swap optimally exchanges all α and β -labelled pixels*; (d) *the α -expansion move optimally selects among current pixel values and the α label*.

[Note: Keep this here or move to Appendix B.6?]

When ρ_p is a quadratic function, the resulting Markov Random Field is called a Gaussian Markov Random Field (GMRF), and its minimum can be found by sparse linear system solving (3.102). When the weighting functions are uniform, the GMRF becomes a special case of Wiener filtering §3.3.1. Allowing the weighting functions to depend on the input image (a special kind of Conditional Random Field, which I describe below) enables quite sophisticated image processing algorithms to be performed, including colorization (Levin *et al.* 2004), interactive tone mapping (Lischinski *et al.* 2006a), natural image matting (Levin *et al.* 2008), and image restoration (Tappen *et al.* 2007).

When ρ_d and/or ρ_p are non-quadratic functions, gradient descent techniques such as non-linear least squares or iteratively re-weighted least-squares can sometimes be used (Appendix A.3). However, if the search space has lots of local minima, as is the case for stereo matching (Barnard 1989, Boykov *et al.* 2001), more sophisticated techniques are required.

The extension of graph cut techniques to multi-valued problems was first proposed by Boykov *et al.* (2001). In their paper, they develop two different algorithms, called the *swap move* and the *expansion move*, which iterate among a series of binary labeling sub-problems to find a good solution (Figure 3.62). (A global solution is generally not achievable, as the problem is provably NP-hard for general energy functions.) Because both these algorithms use a binary MRF optimization inside their inner loop, they are both susceptible to the kind of constraints on the energy functions that occur in the binary labeling case Kolmogorov and Zabih (2004). Appendix B.6 discusses these algorithms in more detail, along with some more recently developed graph-theoretic approaches to this problem (Ishikawa 2003, Veksler 2007).

Another MRF inference technique (remember that the inference of the *maximum a posteriori* state is the same energy minimization) is *belief propagation* (BP). While belief propagation was

originally developed for inference over trees, where it is exact (Pearl 1988), it has more recently been applied to graphs with loops such as Markov Random Fields (Freeman *et al.* 2000, Yedidia *et al.* 2000). In fact, some of the better performing stereo matching algorithms use loopy belief propagation (LBP) to perform their inference (Sun *et al.* 2003). As before, LBP is discussed in more detail in Appendix B.6 as well as the recent comparative survey paper on MRF optimization (Szeliski *et al.* 2008c).

Figure 3.61 shows an example of image denoising and inpainting (hole filling) using a non-quadratic energy function (non-Gaussian MRF). The original image has been corrupted by noise and a portion of the data (black bar) has been removed. In this case, the loopy belief propagation algorithm computes a slightly lower energy and also a smoother image than the alpha-expansion graph cut algorithm. [Note: Could also show a result of using quadratic energy, for comparison, which is less edge-preserving.]

Of course, the above formula (3.112) for the smoothness term $E_p(i, j)$ just shows the simplest case. In more recent work, Roth and Black (2005) propose a *Field of Experts* (FoE) model, which sums up a large number of exponentiated local filter outputs to arrive at the smoothness penalty. Weiss and Freeman (2007) analyze this approach and compare it to the simpler hyper-Laplacian model of natural image statistics.

Unordered labels

The other case of multi-valued labels where Markov Random Fields are applied are *unordered labels*, i.e., labels where there is no semantic meaning to the difference between two labels values. For example, if we are doing terrain classification from aerial imagery, it makes no sense to take the numeric difference between the labels assigned to forest, field, water, and pavement. In fact, the adjacencies of these various kinds of terrain each have different likelihoods, so it makes more sense to use a prior of the form

$$E_p(i, j) = s_x(i, j)V(l(i, j), l(i + 1, j)) + s_y(i, j)V(l(i, j), l(i, j + 1)), \quad (3.114)$$

where $V(l_0, l_1)$ is a general *compatibility* or *potential* function. (Note that I have also replaced $f(i, j)$ with $l(i, j)$ to make it clearer that these are labels rather than discrete function samples.) An alternative way to write this prior energy (Boykov *et al.* 2001, Szeliski *et al.* 2008c) is

$$E_p = \sum_{(p,q) \in \mathcal{N}} V_{p,q}(l_p, l_q), \quad (3.115)$$

where the (p, q) are neighboring pixels and a spatially varying potential function $V_{p,q}$ is evaluated for each neighboring pair.

An important application of unordered MRF labeling is seam finding in image compositing (Davis 1998, Agarwala *et al.* 2004) (see Figure 3.63, which is explained in more detail in §8.3.2).



Figure 3.63: *An example of an unordered label MRF. Strokes in each of the source images on the left are used as constraints on an MRF optimization, which is solved using graph cuts. The resulting multi-valued label field is shown as a color overlay in the middle image, and the final composite is shown on the right. (Copied, with permission, from (Agarwala et al. 2004)).*

Here, the compatibility $V_{p,q}(l_p, l_q)$ measures the quality of the visual appearance that would result from placing a pixel p from image l_p next to a pixel q from image l_q . As for all MRFs, we assume that $V_{p,q}(l, l) = 0$, i.e., it is perfectly fine to choose contiguous pixels from the same image. For different labels, however, the compatibility compares the pixel values $I_{l_p}(p)$ and $I_{l_q}(q)$ may not be equal.

Consider for example where one image I_0 is all sky blue, i.e., $I_0(p) = I_0(q) = B$, while the other image I_1 has a transition from sky blue, $I_1(p) = B$, to forest green, $I_1(q) = G$. [Note: Not sure if it's worth to add a figure here.] In this case, $V_{p,q}(1, 0) = 0$ (the colors agree), while $V_{p,q}(0, 1) > 0$ (the colors disagree). Therefore, as with terrain classification, this function may not satisfy the kinds of *metric constraints* required to apply the usual swap or expansion move algorithms (Kolmogorov and Zabih 2004).

Conditional random fields

In a classic Bayesian model (3.105–3.107),

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}), \quad (3.116)$$

the prior distribution $p(\mathbf{x})$ is independent of the observations \mathbf{y} . Sometimes, however, it is useful to modify our prior assumptions, say about the smoothness of the field we are trying to estimate, in response to the sensed data. (Whether this makes sense from a probability viewpoint is something I will discuss below, once I've explained the new model.)

Consider the interactive image segmentation problem shown in Figure 3.64 (Boykov and Funka-Lea 2006). In this application, the user draws foreground (red) and background (blue) strokes, and the system then solves a binary MRF labeling problem to estimate the extent of the foreground ob-

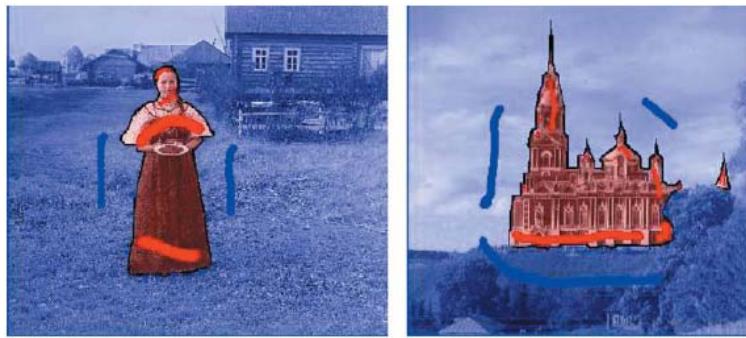


Figure 3.64: *Image segmentation example.* The user draws a few red strokes in the foreground object and a few blue ones in the background. The system computes color distributions for the foreground and background and solves a binary MRF (Boykov and Funka-Lea 2006). The smoothness weights are modulated by the intensity gradients (edges), which makes this a conditional random field (CRF).

[Note: If I can't get permission, use the Grab Cut example in Figure 4.77.]

ject. In addition to minimizing a data term, which measures the pointwise similarity between pixel colors and the inferred region distributions §4.5.4, the MRF is modified so that the smoothness terms $s_x(x, y)$ and $s_y(x, y)$ in Figure 3.59 and (3.112) depend on the magnitude of the gradient between adjacent pixels.

Since the smoothness term now depends on the data, Bayes' Rule (3.116) no longer applies. Instead, we use a direct model for the posterior distribution $p(\mathbf{x}|\mathbf{y})$, whose negative log likelihood can be written as

$$\begin{aligned} E(\mathbf{x}|\mathbf{y}) &= E_d(\mathbf{x}, \mathbf{y}) + E_s(\mathbf{x}, \mathbf{y}) \\ &= \sum_p V_p(x_p, \mathbf{y}) + \sum_{(p,q) \in \mathcal{N}} V_{p,q}(x_p, x_q, \mathbf{y}), \end{aligned} \quad (3.117)$$

using the notation introduced in (3.115). The resulting probability distribution is called a *conditional random field* (CRF), and was first introduced to the computer vision field by Kumar and Hebert (2003b), based on earlier work in text modeling by Lafferty *et al.* (2001).

Figure 3.65 shows a graphical model where the smoothness terms depend on the data values. In this particular model, each smoothness term depends only on its adjacent pair of data values, i.e., terms are of the form $V_{p,q}(x_p, x_q, y_p, y_q)$ in (3.117).

The idea of modifying smoothness terms in response to input data is not new. Boykov and Jolly (2001) used this idea for image segmentation, as shown in Figure 3.64, and it is now widely used in image segmentation §4.5.4 (Blake *et al.* 2004, Rother *et al.* 2004), de-noising (Tappen *et al.* 2007), and object recognition §14.4.3 (Shotton *et al.* 2006, Winn and Shotton 2006, Shotton *et al.* 2008a).

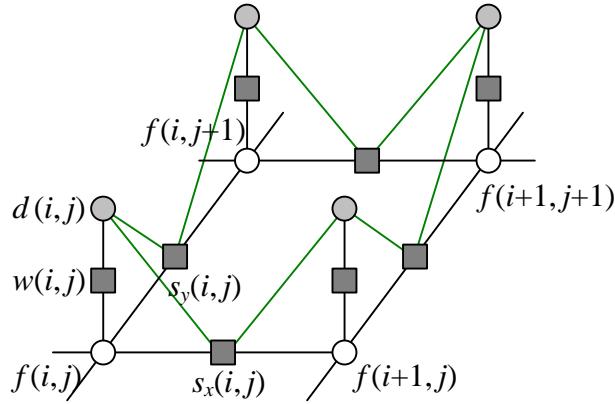


Figure 3.65: *Graphical model for a conditional random field (CRF). The additional green edges show how combinations of sensed data influence the smoothness in the underlying MRF prior model, i.e., $s_x(i,j)$ and $s_y(i,j)$ in (3.112) depend on adjacent $d(i,j)$ values. These additional links (factors) enable the smoothness to depend on the input data. However, they make sampling from this MRF more complex.*

In stereo matching, the idea of encouraging disparity discontinuities to coincide with intensity edges goes back even further to the early days of optimization and MRF-based algorithms (Gamble and Poggio 1987, Fua 1993, Bobick and Intille 1999, Boykov *et al.* 2001) and is discussed in more detail in §10.5.

In addition to using smoothness terms that adapt to the input data, Kumar and Hebert (2003b) also compute a neighborhood function over the input data for each $V_p(x_p, \mathbf{y})$ term, as illustrated in Figure 3.66, instead of using the classic unary MRF data term $V_p(x_p, y_p)$ shown in Figure 3.59.¹⁹ Because such neighborhood functions can be thought of as *discriminant* functions (a term widely used in machine learning (Bishop 2006)), they call the resulting graphical model a *discriminative random fields* (DRF). In their paper, Kumar and Hebert (2006) show that DRFs outperform similar CRFs on a number of applications such as structure detection (Figure 3.67) and binary image denoising.

Here again, one could argue that previous stereo correspondence algorithms also look at a neighborhood of input data, either explicitly, because they compute correlation measures (Crimini *et al.* 2006) as data terms, or implicitly, because even pixel-wise disparity costs look at several pixels in either the left or right image (Barnard 1989, Boykov *et al.* 2001).

[Note: Is it worth discussing (Kumar and Hebert 2003a), which talks about DRFs and natural image statistics and (Kumar and Hebert 2005), which develops hierarchical DRFs? Re-read and

¹⁹ Kumar and Hebert (2006) call the unary potentials $V_p(x_p, \mathbf{y})$ *association potentials* and the pairwise potentials $V_{p,q}(x_p, y_q, \mathbf{y})$ *interaction potentials*.

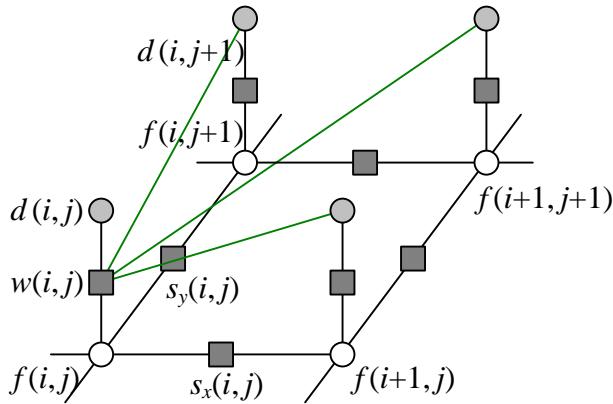


Figure 3.66: *Graphical model for a discriminative random field (DRF). The additional green edges show how combinations of sensed data (e.g., $d(i, j + 1)$ etc.) influence the data term for $f(i, j)$. The generative model is therefore more complex, i.e., we can't just apply a simple function to the unknown variables and add noise.*

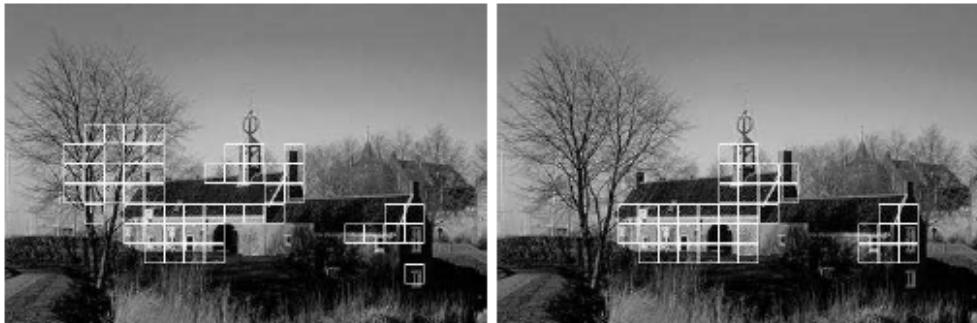


Figure 3.67: *Structure detection results using an MRF (left) and a DRF (right) (Kumar and Hebert 2006).*

decide.]

What, then are the advantages and disadvantages of using conditional or discriminative random fields instead of MRFs?

Classic Bayesian inference (MRF) assumes that the prior distribution of the data is independent of the measurements. This makes a lot of sense: if you see a pair of snake eyes on your first throw at craps, it would be unwise to assume that they will always show up thereafter. However, if after playing for a long time you detect a statistically significant bias, you may want to adjust your prior.

What CRFs do, in essence, is to select or modify the prior model based on observed data. This can be viewed as making a partial inference over additional hidden variables or correlations between the unknowns (say a label, depth, or clean image) and the knowns (observed image(s)).

In some cases, the CRF approach makes a lot of sense, and is in fact the only plausible way to proceed. For example, in grayscale image colorization [*Note: not sure where this will end up, but add a cross-reference here*] (Levin *et al.* 2004), the best way to transfer the continuity information from the input grayscale image to the unknown color image is to modify local smoothness constraints.

Similarly, for simultaneous segmentation and recognition (Shotton *et al.* 2006, Winn and Shotton 2006, Shotton *et al.* 2008a), it makes a lot of sense to permit strong color edges to influence the semantic image label continuities.

In other cases, such as image de-noising, the situation is more subtle. Using a non-quadratic (robust) smoothness term as in (3.112) plays a qualitatively similar role to setting the smoothness based on local gradient information in a Gaussian MRF (GMRF) (Tappen *et al.* 2007). [*Note: In more recent work, (Tanaka and Okutomi 2008) use a larger neighborhood and full covariance matrix on a related Gaussian MRF.*] The advantage of the latter, when the smoothness can be correctly inferred, is that the resulting quadratic energy can be minimized in a single step. However, for situations where the discontinuities are not self-evident in the input data, such as for piecewise-smooth sparse data interpolation, (Blake and Zisserman 1987, Terzopoulos 1988), classic robust smoothness energy minimization may be preferable. Thus, as with most computer vision algorithms, a careful analysis of the problem at hand and desired robustness and computation constraints may be required to choose the best technique (i.e., *caveat emptor*).

Perhaps the biggest advantage of CRFs and DRFs, as argued by Kumar and Hebert (2006) and Tappen *et al.* (2007), is that learning the model parameters is sometimes easier (see also (Blake *et al.* 2004)). While learning parameters in MRFs and their variants is not a topic that I cover in this book, interested readers can find more details in recently published articles (Kumar and Hebert 2006, Tappen *et al.* 2007, Tappen 2007).

[*Note: TODO: turn the blurred graphical model into an exercise*] Discuss the MRF corresponding to blurring, which is less tractable than classic MRFs.

[*Note: Not sure if the stuff below is needed any more:*]

Note that (Boykov and Funka-Lea 2006) has a nice (brief) survey of various energy-based techniques related to object extraction (see their Figure 1). Be sure to cite this in §4.5.4 and elsewhere (e.g., §4.4.1 and §4.4.3).

Forward pointer to level sets §11.3.2? These can also be used for image cleanup (read up on it) and hole filling / inpainting (Bertalmio *et al.* 2000, Bertalmio *et al.* 2003, Criminisi *et al.* 2004). Also, how about Korkaram's scratch removal?

Forward pointer to super-resolution §9.3, texture-based techniques §9.5.

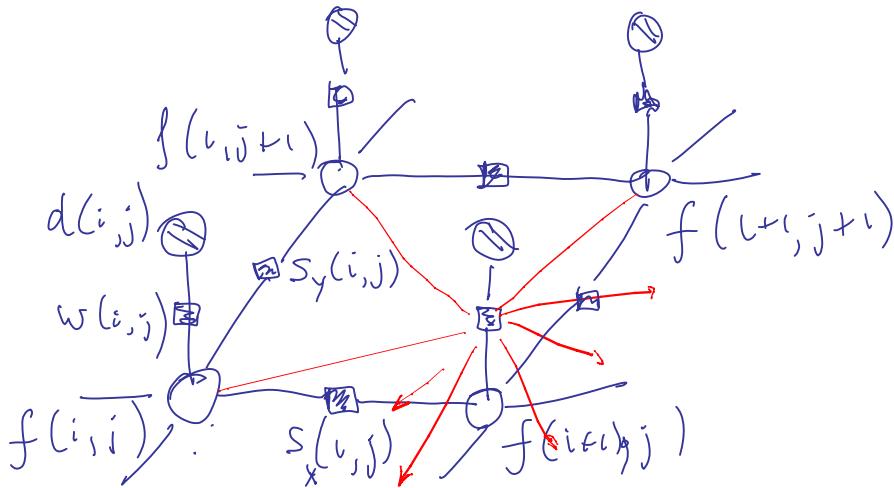


Figure 3.68: *Graphical model for a Markov random field with a more complex measurement model. The additional colored edges show how combinations of unknown values (say in a sharp image) produce the measured values (a noisy blurred image). The resulting graphical model is still a classic MRF and is just as easy to sample from, but some inference algorithms (e.g., graph cut based algorithms) may not be applicable because of the increased network complexity (i.e., state changes during the inference become more entangled, or another way of saying it, the posterior MRF has much larger cliques).*

3.6.3 Application: Image restoration

Do de-noising with Gaussian priors, show equivalence to Wiener filtering using Fourier transform.

Then, do de-noising with non-Gaussian priors (MRF), see if it does better. Some of this may be in the MRF study (actually, I think that was simple inpainting). Mention Steerable Random Fields (Roth and Black 2007), since these may do even better.

Mention the SSIM metric (Wang *et al.* 2004, Wang *et al.* 2005)

3.7 Exercises

Ex 3.1 (Color balance) Write a simple application to change the color balance of an image by multiplying each color value by a different user-specified constant. (If you want to get fancy, you can make this application interactive with sliders.)

1. Do you get different results if you take out the gamma transformation before/after doing the multiplication? Why or why not?
2. Take the same picture with your digital camera using different color balance settings (most

cameras control the color balance from one of the Menus). Can you recover what the color balance ratios are between the different settings? (You may need to put your camera on a tripod and/or align the images manually or automatically to make this work. Alternatively, use a color checker chart.) [Note: add reference to figure or citation on color charts here as well as in *information.tex* and *compphoto.tex*. According to my sleuthing on the Web (2007-11-04) GretagMacbeth was recently acquired by X-Rite <http://www.xrite.com>, which now sells the product under the name “Munsell ColorChecker Chart”.]

3. If you have access to the RAW image for the camera, perform the de-mosaicing yourself (§9.3.1), or downsample the image resolution to get a “true” RGB image. Does your camera perform a simple linear mapping between RAW values and the color-balanced values in a JPEG? (Some high end cameras have a RAW+JPEG mode, which makes this comparison much easier.)
4. Can you think of any reason why you might want to perform a color twist (§3.1.2) on the images? See also Exercise 2.8 for some related ideas.

Ex 3.2 (Compositing and reflections) §3.1.3 describes the process of compositing an alpha-matted image on top of another one. Answer the following questions and optionally validate them experimentally:

1. Most captured images have gamma correction applied to them. Does this invalidate the basic compositing equation (3.8), and if so, show how it should be fixed? [Note: (Blinn 1999, Blinn 2003) has a discussion of this.]
2. The additive (pure reflection) model may have limitations. What happens if the glass is tinted, especially to a non-gray hue? How about if the glass is dirty or smudged? How could you model wavy glass or other kinds of refractive objects? [Note: The answers to most of these questions are in the Environment Matting papers (Zongker et al. 1999, Chuang et al. 2000) and §12.4.]

Ex 3.3 (Blue screen matting) §3.1.3

[Note: Move this to another chapter, e.g., §9.4?]

Ex 3.4 (Difference keying) Implement a difference keying algorithm, e.g., §3.1.3 and (Toyama et al. 1999), consisting of the following steps:

1. Compute the mean and variance (or median and robust variance) at each pixel in an “empty” video sequence.

2. For each new frame, classify each pixel into foreground or background (set the background pixels to $\text{RGBA}=0$).
3. (Optional) Compute the alpha channel, and composite over a new background.
4. (Optional) Clean up the image using morphology ([§3.2.2](#)), label the connected components ([§3.2.5](#)), compute their centroids, and track them from frame to frame. Use this to build a “people counter”.
5. (Optional – remove this) Build a “people remover” through median filtering.

[Note: Can also move this (or forward pointer to) video processing [§13.1](#).]

Ex 3.5 (Photo effects) Write a variety of photo enhancement / effects filters: contrast, solarization (quantization), etc. Which ones are useful (perform sensible corrections), and which ones are more creative (create unusual images)?

Ex 3.6 (Histogram equalization) Compute the gray level (luminance) histogram for an image, and equalize it so that the tones look better (and the image is less sensitive to exposure settings). You may want to use the following steps:

1. Convert the color image to luminance ([§3.1.2](#)).
2. Compute the histogram, the cumulative distribution, and the compensation transfer function ([§3.1.4](#)).
3. (Optional) Try to increase the “punch” in the image by ensuring that a certain fraction of pixels (say 5%) are mapped to pure black and white.
4. (Optional) Limit the local *gain* $f'(I)$ in the transfer function. One way to do this is to limit $f(I) < \gamma I$ and/or $f'(I) < \gamma$ while performing the accumulation ([3.9](#)), keeping any unaccumulated values “in reserve”. (I’ll let you figure out the exact details. :-)
5. Compensate the luminance channel through the lookup table, and re-generate the color image using color ratios ([2.112](#)).
6. (Optional) Color values that are *clipped* in the original image, i.e., have one or more saturated color channels, may appear unnatural when remapped to a non-clipped value. Extend your algorithm to handle this case in some useful way.

Ex 3.7 (Local histogram equalization) Compute the gray level (luminance) histograms for each patch, but add to vertices based on distance (a spline).

1. Build on previous exercise (luminance computation)
2. Distribute values (counts) to adjacent vertices (bilinear)
3. Convert to CDF (look-up functions)
4. Optional low-pass filtering of CDFs
5. Interpolate adjacent CDFs for final lookup

Ex 3.8 (Padding for neighborhood operations) Write down the formulas for computing the padded pixel values $\tilde{f}(i, j)$ as a function of the original pixels values $f(k, l)$ and the image width and height (M, N) for *each* of the padding modes shown in Figure 3.12. For example, for replication (clamping),

$$\tilde{f}(i, j) = f(k, l), \quad \begin{aligned} k &= \max(0, \min(M - 1, i)), \\ l &= \max(0, \min(N - 1, j)), \end{aligned}$$

(Hint: you may want to use the min, max, mod, and absolute value operators in addition to the regular arithmetic operators.)

Describe in more detail the advantages and disadvantages of these various modes.

[Optional] Check what your graphics card does by drawing a texture-mapped rectangle where the texture coordinates lie beyond the $[0.0, 1.0]$ range and using different texture clamping modes.

Ex 3.9 (Separable filters) Implement convolution with a separable kernel. The input should be a grayscale or color image along with the horizontal and vertical kernels. Make sure you support the padding mechanisms developed in the previous exercise. You will need this functionality for some of the later exercises. If you already have access to separable filtering in an image processing package you are using (such as IPL), skip this exercise.

- (Optional) Use Pietro Perona's (1995) technique to approximate convolution as a sum of a number of separable kernels. Let the user specify the number of kernels, and report back some sensible metric of the approximation fidelity.

Ex 3.10 (Discrete Gaussian filters) Discuss the following issues with implementing a discrete Gaussian filter:

- If you just sample the equation of a continuous Gaussian filter at discrete locations, will you get the desired properties, e.g., will the coefficients sum up to 0? Similarly, if you sample a derivative of a Gaussian, do the samples sum up to 0 and/or have vanishing higher order moments?

- Would it be preferable to take the original signal, interpolate it with a sinc, blur with a continuous Gaussian, then pre-filter with a sinc before re-sampling? Is there a simpler way to do this in the frequency domain?
- Alternatively, would it make more sense to produce a Gaussian frequency response in the Fourier domain and to then take an inverse FFT to obtain a discrete filter?
- How does truncation of the filter change its frequency response? Does it introduce any additional artifacts?
- Are the resulting two-dimensional filters as rotationally invariant as their continuous analogues? Is there some way to improve this? In fact, can any 2D discrete (separable or non-separable) filter be truly rotationally invariant?

Ex 3.11 (Sharpening, blur, and noise removal) Implement some softening, sharpening, non-linear diffusion (selective sharpening / noise removal filters, such as Gaussian, median, and bilateral §3.2.2, as discussed in §3.3.2.

Take blurry or noisy images (shooting in low light is a good way to get both) and to try to improve their appearance and legibility.

Ex 3.12 (Steerable filters) Implement Freeman and Adelson's (1991) steerable filter algorithm. The input should be a grayscale or color image, and the output should be a multi-banded image consisting of the following bands:

- $G_1^{0^\circ}$ and $G_1^{90^\circ}$

The coefficients for the filters can be found in (Freeman and Adelson 1991).

Test the various order filters on a number of images of your choice, and see if you can reliably find corner and intersection features. These filters will be quite useful later to detect elongated structures such as lines §4.3.

Ex 3.13 (Distance transform) Implement some (raster-scan) algorithms for city block and Euclidean distance transforms. Can you do it without peeking at the literature (Danielsson 1980, Borgefors 1986)? If so, what problems did you come across and resolve? Hint: for the Euclidean algorithm, you need to keep pairs of values at each pixel indicating the minimum vectorial distance to the boundary.

Later on, you can use the distance functions you compute to perform *feathering* during image stitching §8.3.2.

Ex 3.14 (Connected components) Implement one of the connected component algorithm from §3.2.5 or (Haralick and Shapiro 1992, §2.3), and discuss its computational complexity.

Threshold or quantize an image to obtain a variety of input labels and then compute the area statistics for the regions that you find.

[Note: See what latest Zisserman long-range matching ECCV'2002 paper does (Schaffalitzky and Zisserman 2002).]

Ex 3.15 (Fourier transform) Prove the properties of the Fourier transform listed in Table 3.1, and derive the formulas for the Fourier transforms listed in Tables 3.2 and 3.3. These exercises are very useful if you want to become comfortable with working with Fourier transforms, which is a very useful skill when analyzing and designing the behavior and efficiency of many computer vision algorithms.

Ex 3.16 (Wiener filtering) Estimate the frequency spectrum of your personal photo collection, and use this to perform Wiener filtering on a few images with varying degrees of noise.

- Collect a few hundred of your images by re-scaling them to fit within a 512×512 window and cropping them.
- Take their Fourier transforms, throw away the phase information, and average together all of the spectra.
- Pick two of your favorite images and add varying amounts of Gaussian noise, $\sigma_n \in \{1, 2, 5, 10, 20\}$ gray levels.
- For each image/noise combination, determine by eye which width of a Gaussian blurring filter σ_s gives the best de-noised result. You will have to make a subjective decision between sharpness and noise.
- Compute the Wiener filtered version of all the noised images, and compare these against your hand-tuned Gaussian-smoothed images.
- **[Extra credit]:** Do your image spectra have a lot of energy concentrated along the horizontal and vertical axes ($f_x = 0$ and $f_y = 0$)? Can you think of an explanation for this? Does rotating your image samples by 45° move this energy to the diagonals? If not, is it possible that this is due to edge effects in the Fourier transform? Can you suggest some techniques for reducing such effects? [Possible answers: reflect images top and bottom before taking the transform. Mask the image with a center weight and/or edge rolloff.] [Note: Generate a separate answer key file by having the exercises in separate files and defining a special answer macro.]

Ex 3.17 (Deblurring using Wiener filtering) Use Wiener filtering to deblur some images.

- Modify the Wiener filter derivation (3.65)–(3.73) to incorporate blur (3.74). [*Note: The answer to this question is in my Wiener filter.jnt file.*]
- Discuss the resulting Wiener filter in terms of its noise suppression and frequency boosting characteristics.
- Assuming that the blur kernel is Gaussian and the image spectrum follows an inverse frequency law, compute the frequency response of the Wiener filter and compare it to the unsharp mask.
- Synthetically blur two of your sample images with Gaussian blur kernels of different radii, add noise, and then perform Wiener filtering.
- Repeat the above experiment with a “pillbox” (disc) blurring kernel, which is characteristic of a finite aperture lens (§2.2.3). Compare these results to Gaussian blur kernels (be sure to inspect your frequency plots).
- It has been suggested that regular apertures are anathema to de-blurring because they introduce zeros in the sensed frequency spectrum (Veeraraghavan *et al.* 2007). Show that this is indeed an issue if no prior model is assumed for the signal (i.e., $P_s^{-1}l1$). If a reasonable power spectrum is assumed, is this still a problem (do we still get banding/ringing artifacts)?

Ex 3.18 (High-quality image resampling) Implement several of the low-pass filters presented in §3.4.1 and also the discussion of the windowed sinc shown in Table 3.2 and 3.30. Feel free to implement other filters from (Wolberg 1990) or (Unser 1999).

Apply your filters to continuously resize an image (both magnifying/interpolating and minifying/decimating it), and compare the resulting animations for several filters. Use both a synthetic chirp image like the one shown in Figure 3.69a and natural images with lots of high-frequency detail (Figure 3.69b-c). (These particular images are available on the book website.)

You may find it helpful to write a simple visualization program that continuously plays the animations for two or more filters at once and that lets you “blink” between different results.

Discuss the merits and deficiencies of each filter, as well as its speed vs. quality tradeoff.

Ex 3.19 (Pyramids) Construct an image pyramid. The input should be a grayscale or color image, a separable filter kernel, and the number of desired levels. Implement at least the following kernels:

- 2×2 block filtering
- Burt & Adelson’s binomial kernel $1/16(1, 4, 6, 4, 1)$ (Burt and Adelson 1983a)

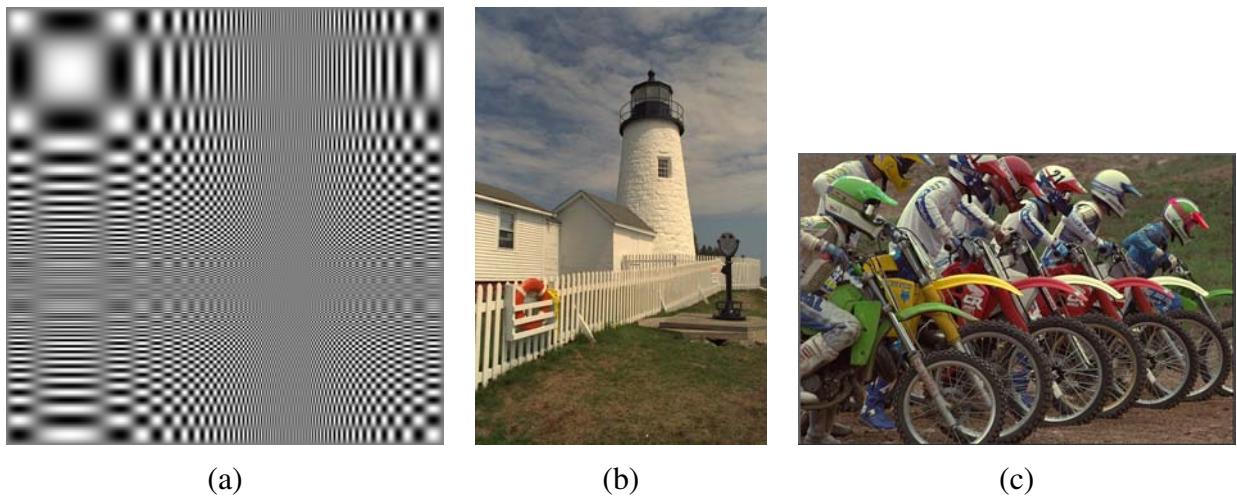


Figure 3.69: *Sample images for testing the quality of resampling algorithms: (a) a synthetic chirp; (b) some high-frequency images from the image compression community.*

- a high-quality 7 or 9-tap filter (using lifting) [Note: Give some recommended coefficients here]; have the students do the lifted version.

Compare the visual quality of the various decimation filters. Also, shift your input image by 1 . . . 4 pixels and compare the resulting decimated (quarter size) image sequence.

Ex 3.20 (Pyramid blending) Write a program that takes as input two color images and a binary mask image and produces the Laplacian pyramid blend of the two images.

1. Construct the Laplacian pyramid for each image.
2. Construct the Gaussian pyramid for the two mask images (the input image and its complement).
3. Multiply each Laplacian image by its corresponding mask and sum the images together (Figure 3.46).
4. Reconstruct the final image from the blended Laplacian pyramid.

Generalize your algorithm to input n images and a label image with values 1 . . . n (the value 0 can be reserved for “no input”). Discuss whether the weighted summation stage (step 3) needs to keep track of the total weight for renormalization, or whether the math just works out. Use your algorithm either to blend two differently exposed image (to avoid under- and over-exposed regions), or to make a creative blend of two different scenes.

Ex 3.21 (Wavelet construction and applications) Implement one of the wavelet families described in §3.4.3 or (Simoncelli and Adelson 1990b), as well as the basic Laplacian pyramid (Exercise 3.19). Apply the resulting representations to one of the following two tasks:

1. **Compression.** Compute the entropy in each band for the different wavelet implementation, assuming a given quantization level (say $\frac{1}{4}$ gray level, to keep rounding error acceptable). Quantize the wavelet coefficients and reconstruct the original images. Which technique performs better? (See (Simoncelli and Adelson 1990b) or any of the multitude of wavelet compression papers for some typical results.)
2. **De-noising.** After computing the wavelets, suppress small values using *coring*. [*Note: Where is this discussed*]. Compare the results of your denoising using different wavelet and/or pyramid representations.

Perform and in-place wavelet transformation (like hierarchical basis transformation), using some simple lifted wavelets. (Specify how many levels to do.) Compute the entropy of the transformed signal (in each “band” separately).

Ex 3.22 (Parametric image warping) Write the code to do affine and perspective image warps (optionally bilinear as well). Try a variety of interpolants, and report on their visual quality. In particular, discuss the following:

- In a MIP-map, selecting only coarser level adjacent to the computed fractional level will produce a blurrier image, while selecting the finer level will lead to aliasing. Explain why this is so, and then discuss whether blending an aliased and blurred image (tri-linear MIP-mapping) is a good idea.
- When the ratio of the horizontal and vertical resampling rates becomes very different (anisotropic), the MIP-map performs even worse. Suggest some approaches to reducing such problems.

Ex 3.23 (Local image warping) Open an image, and deform its appearance. Some possible choices:

1. Click on a number of pixels and move (drag) them to new locations. Interpolate the resulting sparse displacement field to obtain a dense motion field (§3.5.2 and §11.5.1).
2. Draw a number of lines in the image. Move the endpoints of the lines to specify their new positions, and use the Beier-Neely interpolation algorithm (Beier and Neely 1992)(§3.5.2) to get a dense motion field.
3. Overlay a spline control grid, and move one gridpoint at a time (optionally select the level of the deformation).

4. Have a dense per-pixel flow field, and use a soft “paintbrush” with variable size increment/decrement x-y based on mouse-based strokes.
5. [Challenging]: Does the Beier-Neely warp reduce to a sparse point-based deformation as the line segments become shorter (reduce to points)?

Ex 3.24 (Forward warping) Given a displacement field from the previous exercise, write a forward warping algorithm:

1. Write a forward warper using splatting, either nearest neighbor or using soft accumulation §3.5.1.
2. Write a two-pass algorithm, which first forward warps the displacement field, fills in small holes, and then uses inverse warping (*Shade et al. 1998*).
3. Compare the quality to these two algorithms.

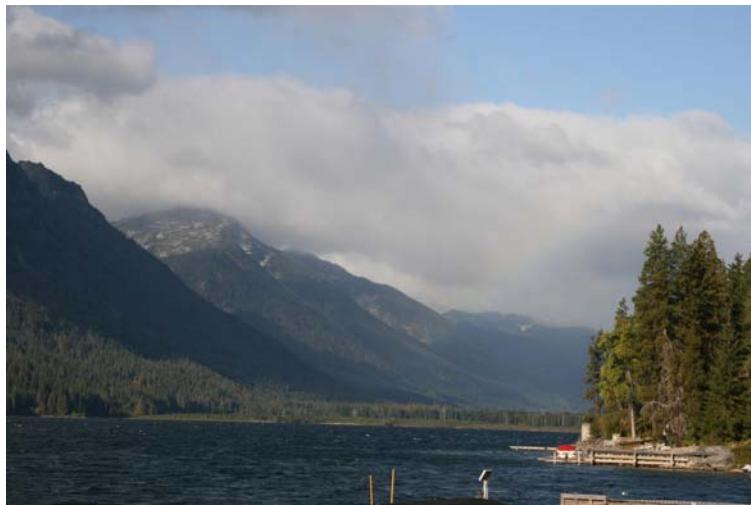
Ex 3.25 (Feature-based morphing) Extend the warping code you wrote in Exercise 3.23 to import two different images and to specify correspondences (point, line, or mesh-based) between the two images.

Create a morph by partially warping the images towards each other and cross-dissolving §3.5.3.

Try using your morphing algorithm to perform an image rotation, and discuss whether it behaves the way you want it to.

Ex 3.26 (2D image editor) Extend the program you wrote in Ex 2.2 to now import images and let you create a “collage” of pictures. You should implement the following steps:

1. Open up a new image (in a separate window).
2. Shift drag (rubber-band) to crop a subregion (or select whole image).
3. Paste into the current canvas.
4. Select the deformation mode (motion model): translation, rigid, similarity, affine, or perspective.
5. Drag any corner of the outline to change its transformation.
6. (optional) Change the relative ordering of the images, and which image is currently being manipulated.



The user should see the composition of the various images pieces on top of each other.

This exercise should be built on the image transformation classes supported in the software library. Persistence of the created representation (save and load) should also be supported (for each image, save its transformation).

Ex 3.27 (3D texture-mapped viewer) Extend the viewer you created in Exercise 2.3 to include texture-mapped polygon rendering. Augment each polygon with (u, v, w) coordinates into an image

Ex 3.28 (Image denoising) Implement at least two of the various image denoising techniques described in this chapter and compare these on both synthetically noised image sequences and on real world (low-light) sequences. Does the performance of the algorithm depend on the correct choice of noise level estimate? Can you draw any conclusions as to which techniques work better?

Ex 3.29 (Rainbow enhancer (challenging)) Take a picture containing a rainbow such as Figure 3.7 and enhance the strength (saturation) of the rainbow.

- Draw an arc in the image delineating the extent of the rainbow.
- Fit an *additive* rainbow function (explain why it's additive) to this arc (it's best to work with linearized pixel values), using the spectrum as the cross section, and estimating the width of the arc and the amount of color being added. This is the trickiest part of the problem, as you need to tease apart the (low-frequency) rainbow pattern and the natural image hiding behind it.
- Amplify the rainbow signal and add it back into the image, re-applying the gamma function if necessary to produce the final image.

Ex 3.30 (Image de-blocking (challenging)) Now that you have some good techniques to distinguish signal from noise, develop a technique to remove the *blocking artifacts* that occur with JPEG at high compression setting §2.3.3. Your technique can be as simple as looking for unexpected edges along block boundaries, to looking at the quantization step as a projection of a convex region of the transform coefficient space onto the corresponding quantized values.

1. Does the knowledge of the compression factor, which is available in the JPEG header information, help you perform better de-blocking?
2. Because the quantization occurs in the DCT transformed YCbCr space (2.111), it may be preferable to perform the analysis in this space. On the other hand, image priors make more sense in an RGB space (or do they?). Decide how you will approach this dichotomy and discuss your choice.
3. While you are at it, since the YCbCr conversion is followed by a chrominance subsampling stage (before the DCT), see if you can restore some of the lost high-frequency chrominance signal using one of the better restoration techniques discussed in this chapter.
4. If your camera has a RAW + JPEG mode, how close can you come to the noise-free true pixel values? (This suggestion may not be that useful, since cameras generally use reasonably high quality settings for their RAW + JPEG models.)

Chapter 4

Feature detection and matching

4.1	Points	209
4.1.1	Feature detectors	212
4.1.2	Feature descriptors	224
4.1.3	Feature matching	228
4.1.4	Feature tracking	237
4.1.5	<i>Application:</i> Performance-driven animation	239
4.2	Edges	240
4.2.1	Edge detection	241
4.2.2	Edge linking	249
4.2.3	<i>Application:</i> Edge editing and enhancement	253
4.3	Lines	255
4.3.1	Successive approximation	255
4.3.2	Hough transforms	256
4.3.3	Vanishing points	261
4.3.4	<i>Application:</i> Rectangle detection	263
4.4	Active contours	264
4.4.1	Snakes	265
4.4.2	Scissors	275
4.4.3	Level Sets	276
4.4.4	<i>Application:</i> Contour tracking and rotoscoping	278
4.5	Region segmentation	278
4.5.1	Split and merge	280
4.5.2	Mean shift and mode finding	284
4.5.3	Normalized cuts	291
4.5.4	Graph cuts and energy-based methods	296
4.5.5	<i>Application:</i> Medical image segmentation	299
4.6	Exercises	300

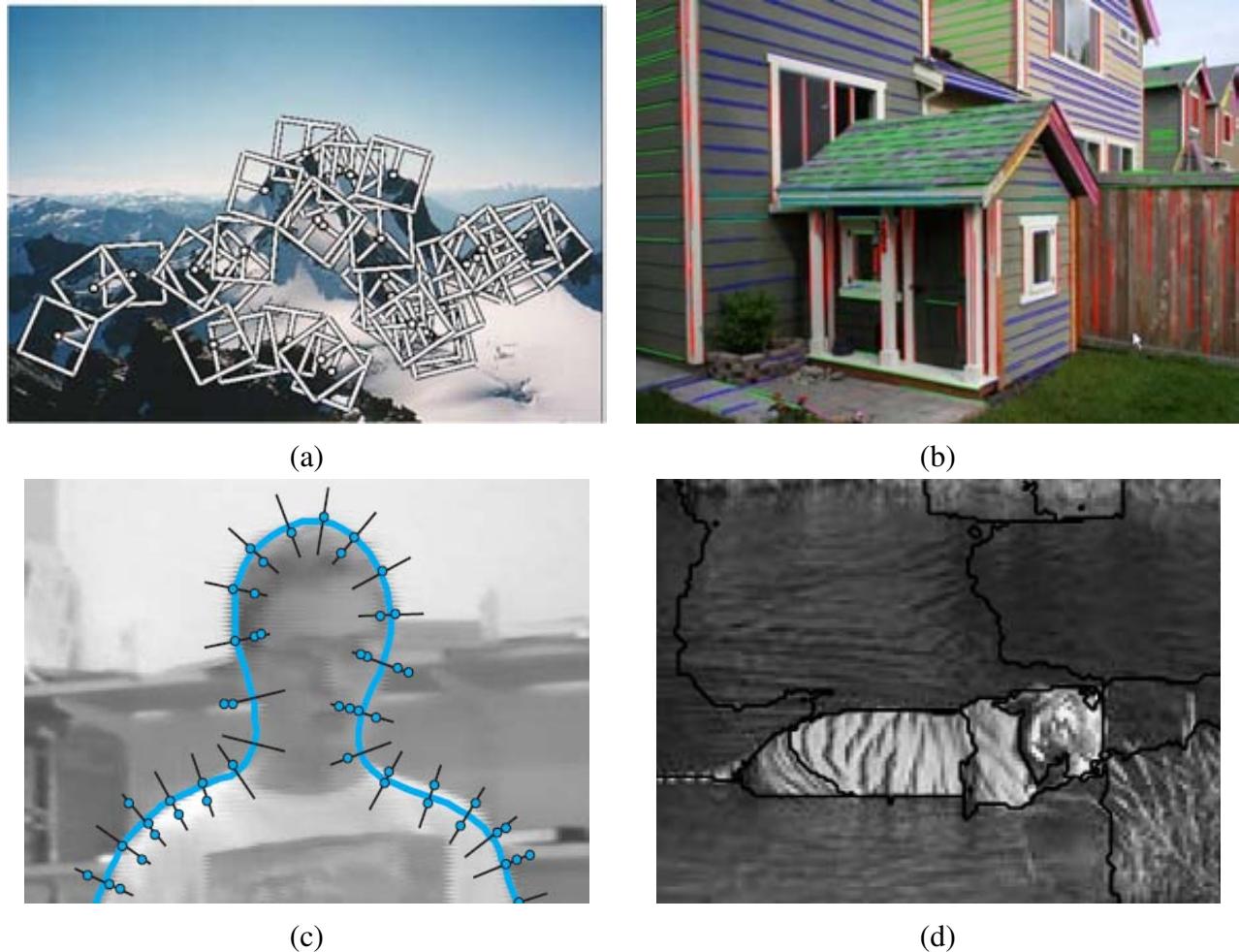


Figure 4.1: A variety of feature detection technique can be used to analyze an image: (a) point-like interest operators (Brown et al. 2005); (b) straight lines (Sinha et al. 2008); (c) active contours (Isard and Blake 1998); (d) region segmentation (Malik et al. 2001).

Feature detection and matching are an essential component of many computer vision applications. Consider the two pairs of images shown in Figure 4.2. For the first pair, we may wish to *align* the two images so that they can be seamlessly stitched into a composite mosaic (§8). For the second pair, we may wish to establish a dense set of *correspondences* so that a 3D model can be constructed or an in-between view could be generated (§10). In either case, what kinds of *features* should you detect and then match in order to establish such an alignment or set of correspondences? Think about this for a few moments before reading on.

The first kind of feature that you may notice are specific locations in the images, such as mountain peaks, building corners, doorways, or interestingly shaped patches of snow. These kinds of localized features are often called *keypoint features* or *interest points* (or even *corners*) and are often described by the appearance of *patches* of pixels surrounding the point location (§4.1). Another class of important features are *edges*, e.g., the profile of the mountains against the sky (§4.2). These kinds of features can be matched based on their orientation and local appearance (*edge profiles*) and can also be good indicators of object boundaries and *occlusion* events in image sequences. Edges can be grouped into longer *curves* and *straight line segments*, which can be directly matched, or analyzed to find *vanishing points* and hence internal and external camera parameters (§4.3). Curves can also be interactively drawn in images and tracked from frame to frame to aid in the *segmentation* of images into constituent parts (§4.4). Finally, large *regions* of uniform color or texture can be extracted as separate entities and used for a variety of correspondence and image manipulation tasks (§4.5). Figure 4.1 shows some examples of these features applied to different images.

In this chapter, I describe some practical approaches to detecting such features and also discuss how feature correspondences can be established across different images. Point features are now used in such a wide variety of applications that I encourage everyone to read and implement some of the algorithms from §4.1. Edges and lines provide information that is complementary to both keypoint and region-based descriptors, and are well-suited to describing object boundaries and man-made objects. These alternative descriptors, while extremely useful, can be skipped in a short introductory course.

4.1 Points

Point features can be used to find a sparse set of corresponding locations in different images, often as a pre-cursor to computing camera pose (§6), which is a prerequisite for computing a denser set of correspondences using stereo matching (§10). Such correspondences can also be used to align different images, e.g., when stitching image mosaics or performing video stabilization (§8). They are also used extensively to perform object instance and category recognition (§14.3-§14.4). A key



Figure 4.2: Two pairs of images to be matched. What kinds of features might one use to establish a set of correspondences between these images?

advantage of keypoints is that they permit matching even in the presence of clutter (occlusion) and large scale and orientation changes.

Feature-based correspondence techniques have been used since the early days of stereo matching (Hannah 1974, Moravec 1983, Hannah 1988) and have more recently gained popularity for image stitching applications (Zoghliami *et al.* 1997, Capel and Zisserman 1998, Cham and Cipolla 1998, Badra *et al.* 1998, McLauchlan and Jaenicke 2002, Brown and Lowe 2003a, Brown *et al.* 2005) as well as fully automated 3D modeling (Beardsley *et al.* 1996, Schaffalitzky and Zisserman 2002, Brown and Lowe 2003b, Snavely *et al.* 2006). [Note: Thin out some of these references?]

There are two main approaches to finding feature points and their correspondences. The first is to find features in one image that can be accurately *tracked* using a local search technique such as correlation or least squares (§4.1.4). The second is to independently detect features in all the images under consideration and then *match* features based on their local appearance (§4.1.3). The former approach is more suitable when images are taken from nearby viewpoints or in rapid succession (e.g., video sequences), while the latter is more suitable when a large amount of motion or appearance change is expected, e.g., in stitching together panoramas (Brown and Lowe 2003a),

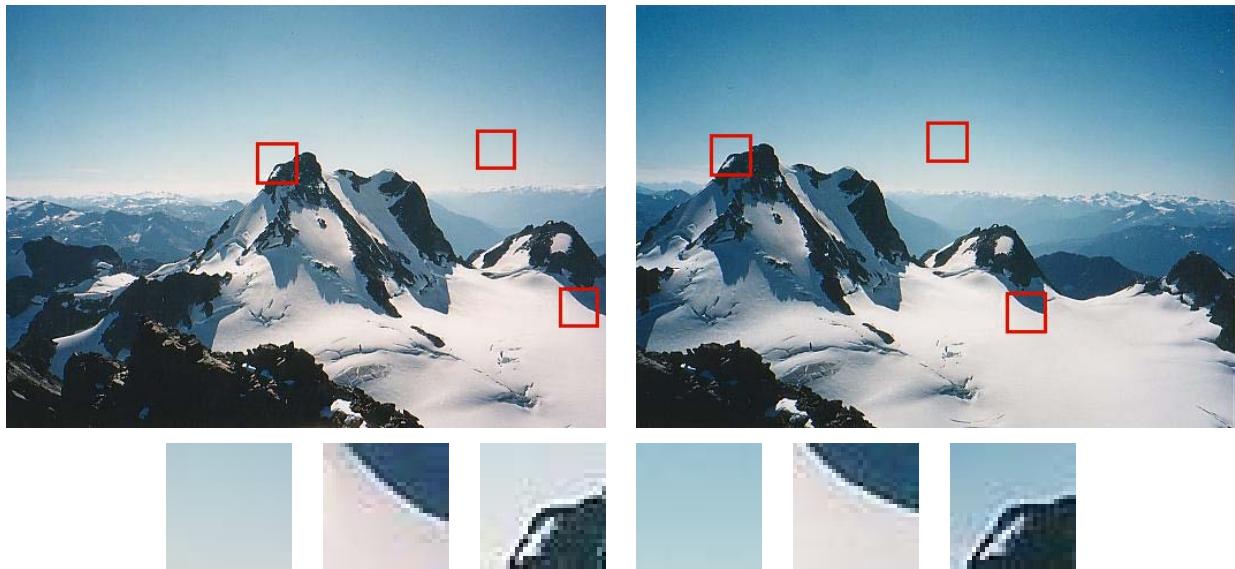


Figure 4.3: *Image pairs with extracted patches below. Notice how some patches can be localized or matched with higher accuracy than others.*

establishing correspondences in *wide baseline stereo* (Schaffalitzky and Zisserman 2002), or performing object recognition (Fergus *et al.* 2003).

In this section, I split the keypoint detection and matching pipeline into four separate stages. During the first *feature detection* (extraction) stage, §4.1.1, each image is searched for locations that are likely to match well in other images. At the second *feature description* stage, §4.1.2, each region around detected keypoint locations is converted into a more compact and stable (invariant) *descriptor* that can be matched against other descriptors. The third *feature matching* stage, §4.1.3, efficiently searches for likely matching candidates in other images. The fourth *feature tracking* stage, §4.1.4, is an alternative to the third stage that only searches a small neighborhood around each detected feature and is therefore more suitable for video processing.

A wonderful example of all of these stages can be found in David Lowe's (2004) *Distinctive image features from scale-invariant keypoints* paper, which describes the development and refinement of his *Scale Invariant Feature Transform* (SIFT). Comprehensive descriptions of alternative techniques can be found in a series of survey and evaluation papers by Schmid, Mikolajczyk, *et al.* covering both feature detection (Schmid *et al.* 2000, Mikolajczyk *et al.* 2005, Tuytelaars and Mikolajczyk 2007) and feature descriptors (Mikolajczyk and Schmid 2005). Shi and Tomasi (1994) and Triggs (2004) also provide nice reviews of feature detection techniques.

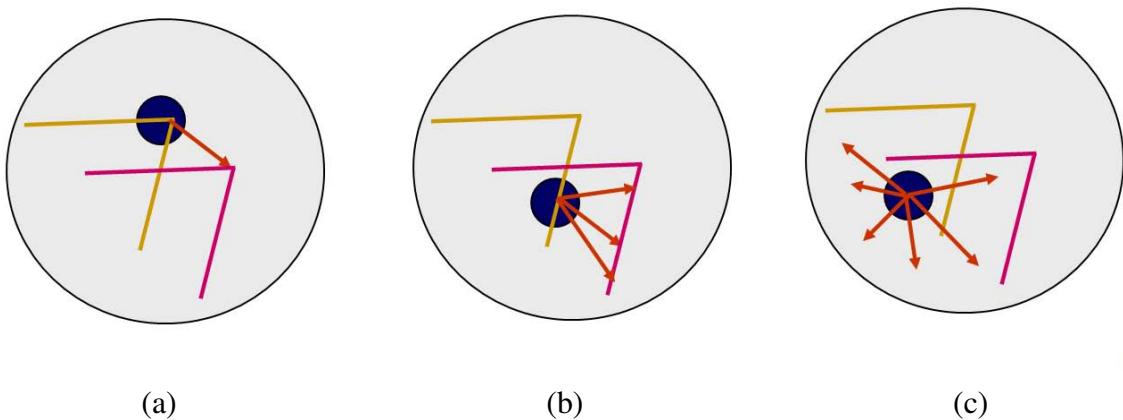


Figure 4.4: *Aperture problems for different image patches:* (a) stable (“corner-like”) flow; (b) classic aperture problem (barber-pole illusion); (c) textureless region. The two images I_0 (yellow) and I_1 (red) are overlaid. The red vector \mathbf{u} indicates the displacement between the patch centers, and the $w(\mathbf{x}_i)$ weighting function (patch window) is shown as a dark circle.

4.1.1 Feature detectors

How can we find image locations where we can reliably find correspondences with other images, i.e., what are *good features to track* (Shi and Tomasi 1994, Triggs 2004)? Look again at the image pair shown in Figure 4.3 and at the three sample *patches* to see how well they might be matched or tracked. As you may notice, textureless patches are nearly impossible to localize. Patches with large contrast changes (gradients) are easier to localize, although straight line segments at a single orientation suffer from the *aperture problem* (Horn and Schunck 1981, Lucas and Kanade 1981, Anandan 1989), i.e., it is only possible to align the patches along the direction *normal* to the edge direction (Figure 4.4b). Patches with gradients in at least two (significantly) different orientations are the easiest to localize, as shown schematically in Figure 4.4a.

These intuitions can be formalized by looking at the simplest possible matching criterion for comparing two image patches, i.e., their (weighted) summed square difference,

$$E_{\text{WSSD}}(\mathbf{u}) = \sum_i w(\mathbf{x}_i)[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2, \quad (4.1)$$

where I_0 and I_1 are the two images being compared, $\mathbf{u} = (u, v)$ is the *displacement* vector, and $w(\mathbf{x})$ is a spatially varying weighting (or window) function. (Note that this is the same formulation I later use to estimate motion between complete *images* §7.1, and that this section shares some material with that later section.)

When performing feature detection, we do not know which other image location(s) the feature will end up being matched against. Therefore, we can only compute how stable this metric is with

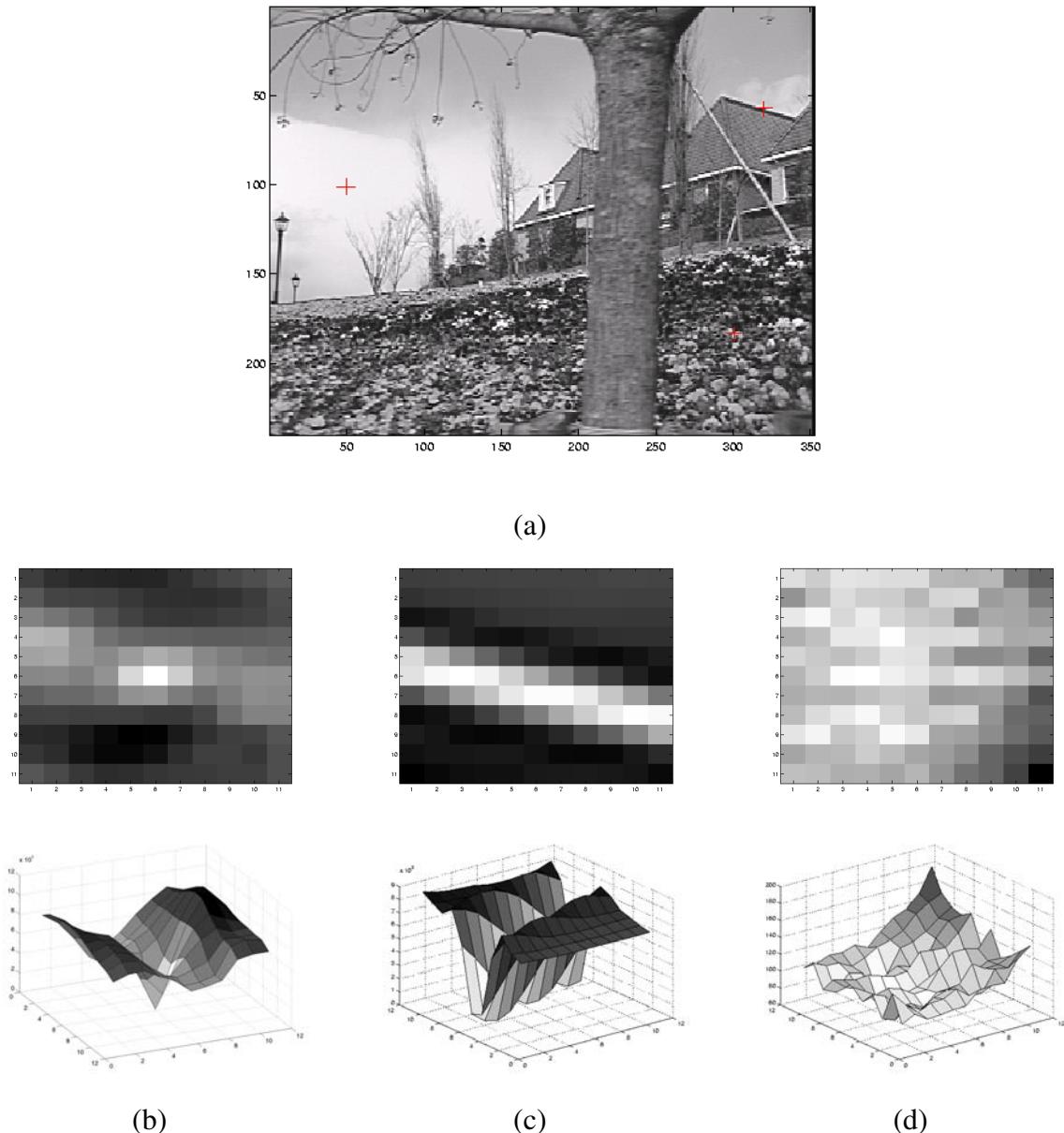


Figure 4.5: *Three different auto-correlation surfaces (b–d) shown as both grayscale images and surface plots. The original image (a) is marked with three red crosses to denote where these auto-correlation surfaces were computed. Patch (b) is from the flower bed (good unique minimum), patch (c) is from the roof edge (one-dimensional aperture problem), and patch (d) is from the cloud (no good peak).*

respect to small variations in position $\Delta\mathbf{u}$ by comparing an image patch against itself, which is known as an *auto-correlation function* or *surface*

$$E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i) [I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2 \quad (4.2)$$

(Figure 4.5).¹ Note how the auto-correlation surface for the textured flower bed (Figure 4.5b, red cross in the lower right-hand quadrant of Figure 4.5a) exhibits a strong minimum, indicating that it can be well localized. The correlation surface corresponding to the roof edge (Figure 4.5c) has a strong ambiguity along one direction, while the correlation surface corresponding to the cloud region (Figure 4.5d) has no stable minimum.

Using a Taylor Series expansion of the image function $I_0(\mathbf{x}_i + \Delta\mathbf{u}) \approx I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}$ (Lucas and Kanade 1981, Shi and Tomasi 1994), we can approximate the auto-correlation surface as

$$E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i) [I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2 \quad (4.3)$$

$$\approx \sum_i w(\mathbf{x}_i) [I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u} - I_0(\mathbf{x}_i)]^2 \quad (4.4)$$

$$= \sum_i w(\mathbf{x}_i) [\nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}]^2 \quad (4.5)$$

$$= \Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u}, \quad (4.6)$$

where

$$\nabla I_0(\mathbf{x}_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y} \right)(\mathbf{x}_i) \quad (4.7)$$

is the *image gradient* at \mathbf{x}_i . This gradient can be computed using a variety of techniques (Schmid *et al.* 2000). The classic “Harris” detector (Harris and Stephens 1988) uses a [-2 -1 0 1 2] filter, but more modern variants (Schmid *et al.* 2000, Triggs 2004) convolve the image with horizontal and vertical derivatives of a Gaussian (typically with $\sigma = 1$). [Note: Bill Triggs doubts that Harris and Stephens (1988) used such a bad filter kernel, as reported in (Schmid *et al.* 2000), but the original publication is hard to find.]

The auto-correlation matrix \mathbf{A} can be written as

$$\mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \quad (4.8)$$

where we have replaced the weighted summations with discrete convolutions with the weighting kernel w . This matrix can be interpreted as tensor (multiband) image, where the outer products of

¹ Strictly speaking, the auto-correlation is the *product* of the two weighted patches; I’m using the term here in a more qualitative sense. The weighted sum of squared differences is often called an *SSD surface* §7.1.

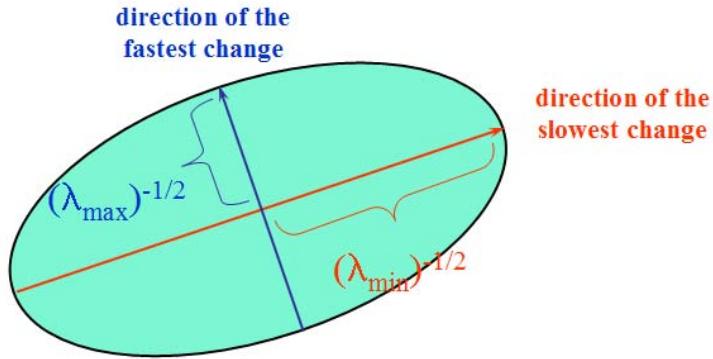


Figure 4.6: *Uncertainty ellipse corresponding to an eigenvalue analysis of the auto-correlation matrix \mathbf{A} .*

[Note: Update this figure for better colors, replacing λ_{\min} with λ_0 and generating a true PDF figure.]

the gradients ∇I are convolved with a weighting function w to provide a per-pixel estimate of the local (quadratic) shape of the auto-correlation function. In more detail, the computation of image that contains a 2×2 matrix \mathbf{A} at each pixel can be performed in two steps:

1. At each pixel, compute the gradient $\nabla I = [I_x \ I_y]$ and then compute the four values $[I_x^2 \ I_x I_y \ I_x I_y \ I_y^2]$;
2. Convolve the resulting 4-band image with a blur kernel w .

As first shown by Anandan (1984, 1989) and further discussed in §7.1.3 and (7.42), the inverse of the matrix \mathbf{A} provides a lower bound on the uncertainty in the location of a matching patch. It is therefore a useful indicator of which patches can be reliably matched. The easiest way to visualize and reason about this uncertainty is to perform an eigenvalue analysis of the auto-correlation matrix \mathbf{A} , which produces two eigenvalues (λ_0, λ_1) and two eigenvector directions (Figure 4.6). Since the larger uncertainty depends on the smaller eigenvalue, i.e., $\lambda_0^{-1/2}$, it makes sense to find maxima in the smaller eigenvalue to locate *good features to track* (Shi and Tomasi 1994).

Förstner-Harris. While Anandan as well as Lucas and Kanade (1981) were the first to analyze the uncertainty structure of the auto-correlation matrix, they did so in the context of associating certainties with optic flow measurements. Förstner (1986) and Harris and Stephens (1988) were the first to propose using local maxima in rotationally invariant scalar measures derived from the auto-correlation matrix to locate keypoints for the purpose of sparse feature matching. (See (Schmid *et al.* 2000, Triggs 2004) for more detailed historical reviews of feature detection algorithms) Both

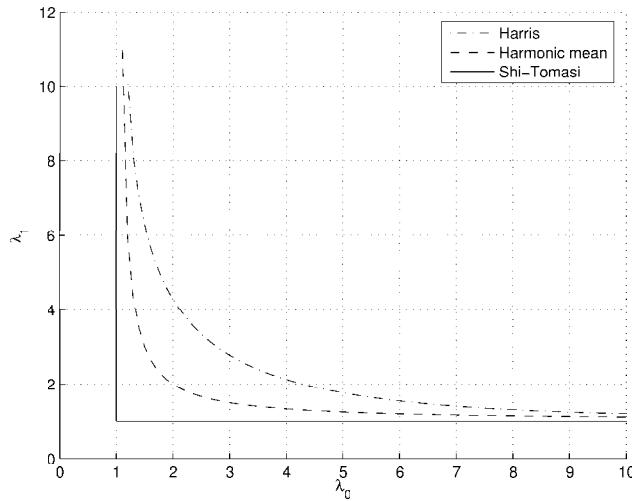


Figure 4.7: Isocontours of popular keypoint detection functions (Brown et al. 2004). Each detector looks for points where the eigenvalues λ_0, λ_1 of $\mathbf{A} = w * \nabla I \nabla I^T$ are both large.

of these techniques also proposed using a Gaussian weighting window instead of the previously used square patches, which makes the detector response insensitive to in-plane image rotations. [Note: Decide if (Förstner 1994) is worth citing as well.]

The minimum eigenvalue λ_0 (Shi and Tomasi 1994) is not the only quantity that can be used to find keypoints. A simpler quantity, proposed by Harris and Stephens (1988) is

$$\det(\mathbf{A}) - \alpha \operatorname{trace}(\mathbf{A})^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2 \quad (4.9)$$

with $\alpha = 0.06$. [Note: Brown et al. (2005) say that $\alpha = 0.04$. Check out the original paper.] Unlike eigenvalue analysis, this quantity does not require the use of square roots, and yet is still rotationally invariant and also downweights edge-like features where $\lambda_1 \gg \lambda_0$. Triggs (2004) suggest using the quantity

$$\lambda_0 - \alpha \lambda_1 \quad (4.10)$$

(say with $\alpha = 0.05$), which also reduces the response at 1D edges, where aliasing errors sometimes inflate the smaller eigenvalue. He also shows how the basic 2×2 Hessian can be extended to parametric motions to detect points that are also accurately localizable in scale and rotation. Brown et al. (2005), on the other hand, use the harmonic mean,

$$\frac{\det \mathbf{A}}{\operatorname{tr} \mathbf{A}} = \frac{\lambda_0 \lambda_1}{\lambda_0 + \lambda_1}, \quad (4.11)$$

which is a smoother function in the region where $\lambda_0 \approx \lambda_1$. Figure 4.7 shows isocontours of the various interest point operators (note that all the detectors require both eigenvalues to be large).

The steps in the basic auto-correlation-based keypoint detector can therefore be summarized as follows:

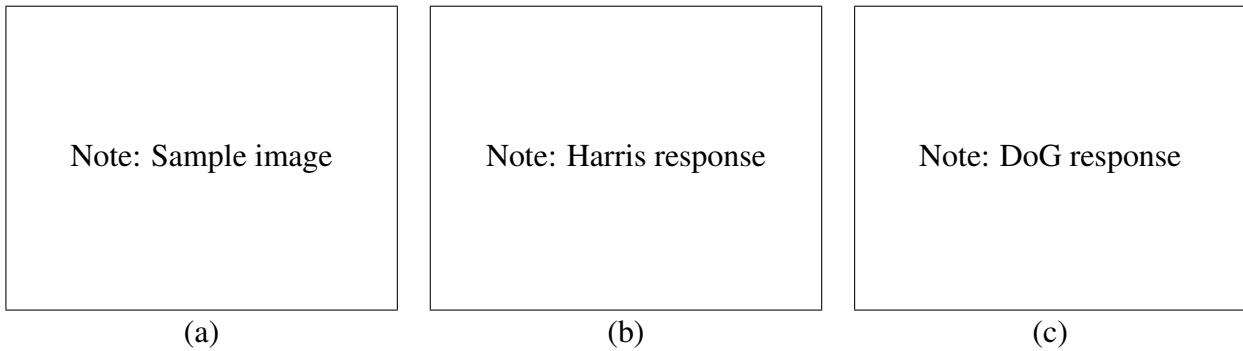


Figure 4.8: *Sample image (a) and two different interest operator responses: (b) Harris; (c) DoG. Local maxima are shown as red dots.*

[Note: Need to generate this figure.]

1. Compute the horizontal and vertical derivatives of the image I_x and I_y by convolving the original image with derivatives of Gaussians §3.2.1.
2. Compute the three images corresponding to the outer products of these gradients (the matrix A is symmetric, so only three entries are needed).
3. Convolve each of these images with a larger Gaussian.
4. Compute a scalar interest measure using one of the formulas discussed above.
5. Find local maxima above a certain threshold and report these as detected feature point locations.

Figure 4.8 shows the resulting interest operator responses for the classic Harris detector as well as the DoG detector discussed below.

Adaptive non-maximal suppression While most feature detectors simply look for local maxima in the interest function, this can lead to an uneven distribution of feature points across the image, e.g., points will be denser in regions of higher contrast. To mitigate this problem, Brown *et al.* (2005) only detect features that are both local maxima and whose response value is significantly (10%) greater than that of all of its neighbors within a radius r (Figure 4.9c–d). They devise an efficient way to associate suppression radii with all local maxima by first sorting all local maxima by their response strength, and then creating a second list sorted by decreasing suppression radius (see (Brown *et al.* 2005) for details). A qualitative comparison of selecting the top n features vs. ANMS is shown in Figure 4.9.

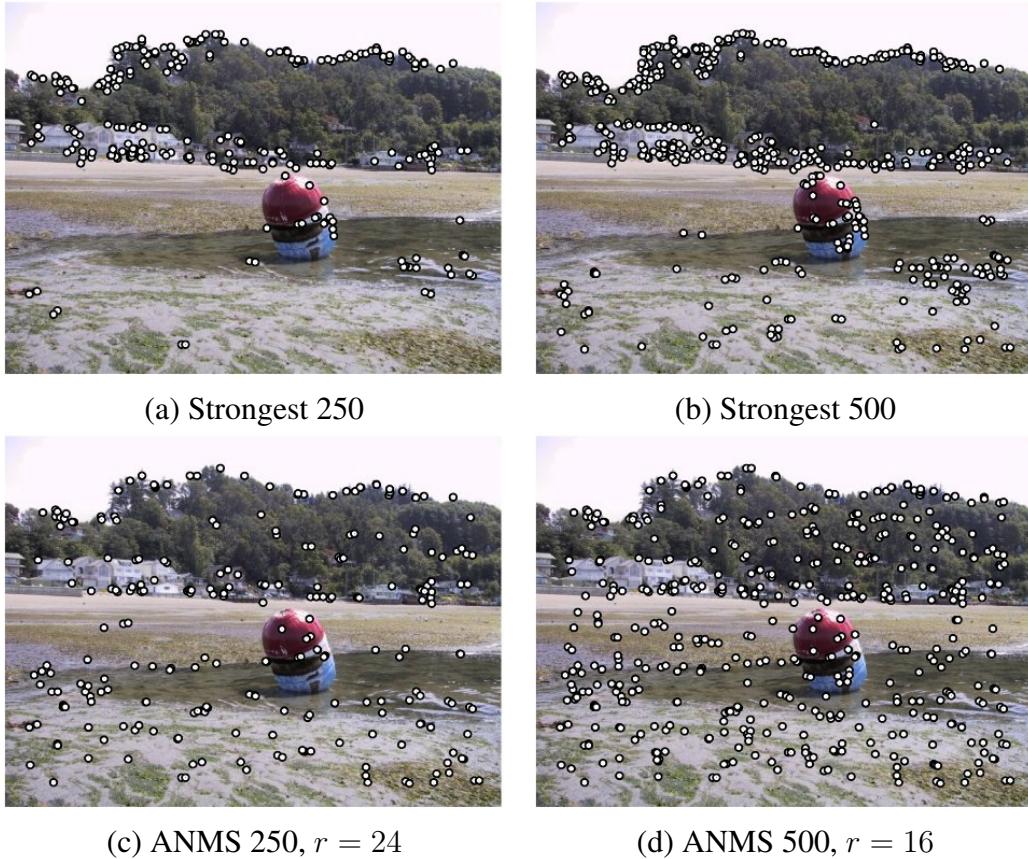


Figure 4.9: *Adaptive non-maximal suppression (ANMS)* (Brown et al. 2005). The two upper images show the strongest 250 and 500 interest points, while the lower two images show the interest points selected with adaptive non-maximal suppression (along with the corresponding suppression radius r). Note how the latter features have a much more uniform spatial distribution across the image.

Measuring repeatability Given the large number of feature detectors that have been developed in computer vision, how can we decide which ones to use? Schmid *et al.* (2000) were the first to propose measuring the *repeatability* of feature detectors, which is defined as the frequency with which keypoints detected in one image are found within ϵ (say $\epsilon = 1.5$) pixels of the corresponding location in a transformed image. In their paper, they transform their planar images by applying rotations, scale changes, illumination changes, viewpoint changes, and adding noise. They also measure the *information content* available at each detected feature point, which they define as the entropy of a set of rotationally invariant local grayscale descriptors. Among the techniques they survey, they find that the improved (Gaussian derivative) version of the Harris operator with $\sigma_d = 1$ (scale of the derivative Gaussian) and $\sigma_i = 2$ (scale of the integration Gaussian) works best.

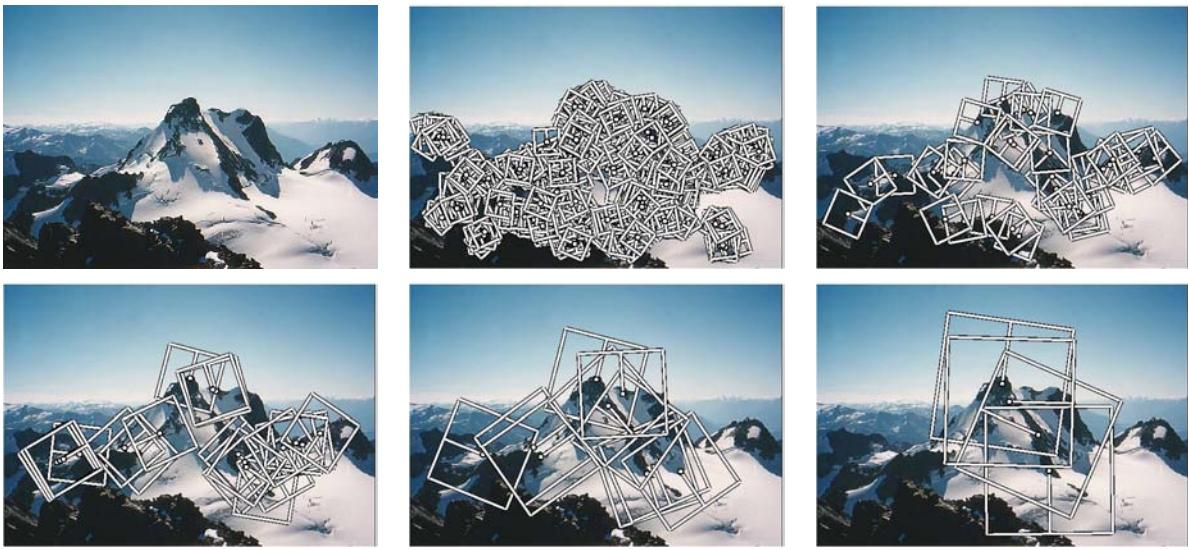


Figure 4.10: *Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels (Brown et al. 2004).* The boxes show the feature orientation and the region from which the descriptor vectors are sampled.

Scale invariance

In many situations, detecting features at the finest stable scale possible may not be appropriate. For example, when matching images with little high frequency (e.g., clouds), fine-scale features may not exist.

One solution to the problem is to extract features at a variety of scales, e.g., by performing the same operations at multiple resolutions in a pyramid and then matching features at the same level. This kind of approach is suitable when the images being matched do not undergo large scale changes, e.g., when matching successive aerial images taken from an airplane, or stitching panoramas taken with a fixed focal length camera. Figure 4.10 shows the output of one such approach, the multi-scale oriented patch detector of Brown *et al.* (2005), for which responses at 5 different scales are shown.

However, for most object recognition applications, the scale of the object in the image is unknown. Instead of extracting features at many different scales and then matching all of these, it is more efficient to extract features that are stable in both location *and* scale (Lowe 2004, Mikolajczyk and Schmid 2004).

Early investigations into scale selection were performed by Lindeberg (1993, 1998b), who first proposed using extrema in the Laplacian of Gaussian (LoG) function as interest point locations. Based on this work, Lowe (2004) proposed computing a set of sub-octave Difference of Gaussian filters (Figure 4.11a), looking for 3-D (space+scale) maxima in the resulting structure

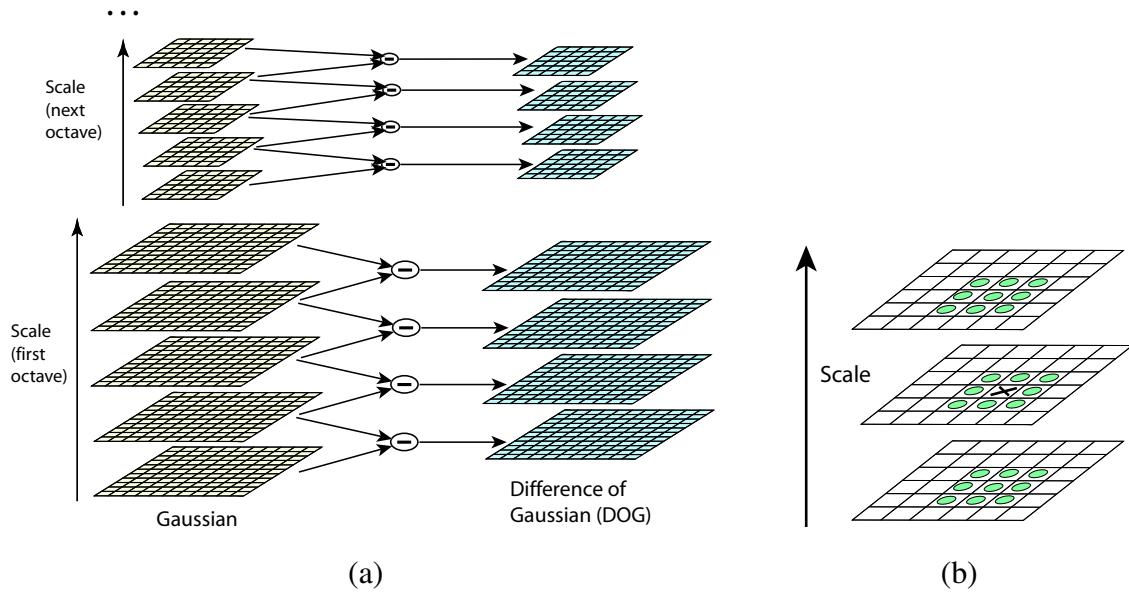


Figure 4.11: Scale-space feature detection using a sub-octave Difference of Gaussian pyramid (Lowe 2004). (a) Adjacent levels of a sub-octave Gaussian pyramid are subtracted to produce Difference of Gaussian images. (b) Extrema (maxima and minima) in the resulting 3D volume are detected by comparing a pixel to its 26 neighbors.

(Figure 4.11), and then computing a sub-pixel space+scale location using a quadratic fit (Brown and Lowe 2002). The number of sub-octave levels was chosen after careful empirical investigation, and was determined to be 3, which corresponds to a quarter-octave pyramid (the same as used by Triggs (2004)).

As with the Harris operator, pixels where there is strong asymmetry in the local curvature of the indicator function (in this case the DoG) are rejected. This is implemented by first computing the local Hessian of the difference image D ,

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}, \quad (4.12)$$

and then rejecting keypoints for which

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} > 10. \quad (4.13)$$

While Lowe's Scale Invariant Feature Transform (SIFT) performs well in practice, it is not based on the same theoretical foundation of maximum spatial stability as the auto-correlation-based detectors. (In fact, its detection locations are often complementary to those produced by such techniques and can therefore be used in conjunction with these other approaches.) In order

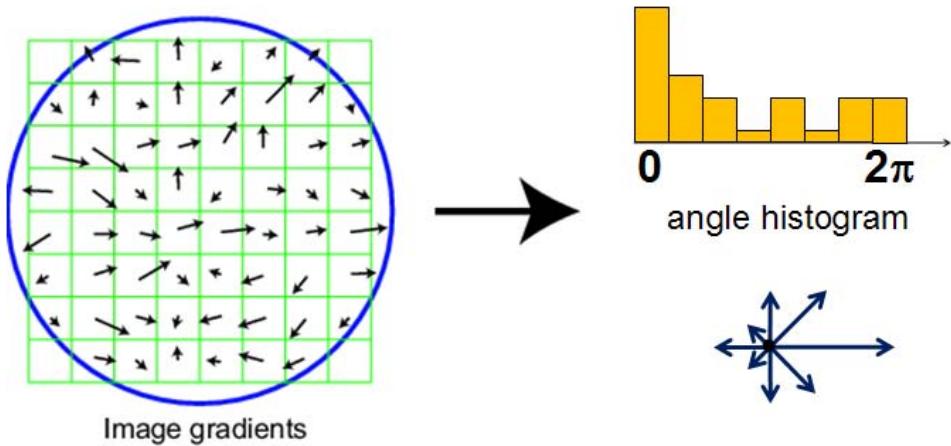


Figure 4.12: A dominant orientation estimate can be computed by creating a histogram of all the gradient orientations (weighted by their magnitudes and/or after thresholding out small gradients), and then finding the significant peaks in this distribution (Lowe 2004).

to add a scale selection mechanism to the Harris corner detector, Mikolajczyk and Schmid (2004) evaluate the Laplacian of a Gaussian function at each detected Harris point (in a multi-scale pyramid) and keep only those points for which the Laplacian is extremal (larger or smaller than both its coarser and finer-level values). An optional iterative refinement for both scale and position is also proposed and evaluated. Additional examples of scale invariant region detectors can be found in (Mikolajczyk *et al.* 2005, Tuytelaars and Mikolajczyk 2007).

Rotational invariance and orientation estimation

In addition to dealing with scale changes, most image matching and object recognition algorithms need to deal with (at least) in-plane image rotation. One way to deal with this problem is to design descriptors that are rotationally invariant (Schmid and Mohr 1997), but such descriptors have poor discriminability, i.e. they map different looking patches to the same descriptor.

A better method is to estimate a *dominant orientation* at each detected keypoint. Once the local orientation and scale of a keypoint have been estimated, a scaled and oriented patch around the detected point can be extracted and used to form a feature descriptor (Figures 4.10 and 4.17).

The simplest possible orientation estimate is the average gradient within a region around the keypoint. If a Gaussian weighting function is used (Brown *et al.* 2005), this average gradient is equivalent to a first order steerable filter §3.2.1, i.e., it can be computed using an image convolution with the horizontal and vertical derivatives of Gaussian filter (Freeman and Adelson 1991). In order to make this estimate more reliable, it is usually preferable to use a larger aggregation window

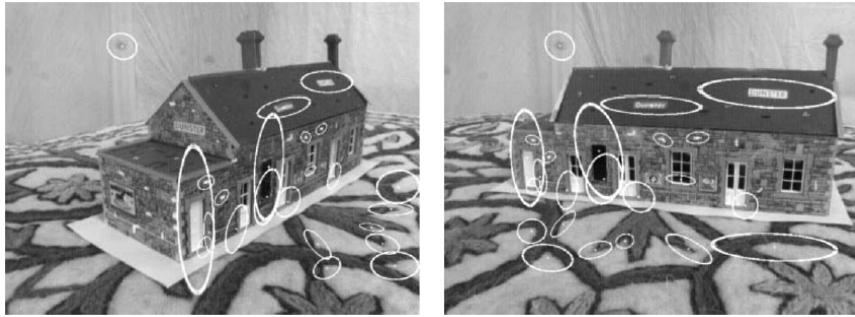


Figure 4.13: *An example of using affine region detectors to match two images taken from dramatically different viewpoints (Mikolajczyk and Schmid 2004).*

(Gaussian kernel size) than the detection window size (Brown *et al.* 2005). The orientations of the square boxes shown in Figure 4.10 were computed using this technique.

Sometime, however, the averaged (signed) gradient in a region can be small and therefore an unreliable indicator of orientation. A better technique in this case is to look at the *histogram* of orientations computed around the keypoint. Lowe (2004) computes a 36-bin histogram of edge orientations weighted by both gradient magnitude and Gaussian distance to the center, finds all peaks within 80% of the global maximum, and then computes a more accurate orientation estimate using a 3-bin parabolic fit (Figure 4.12).

Affine invariance

While scale and rotation invariance are highly desirable, for many applications such as *wide baseline stereo matching* (Pritchett and Zisserman 1998, Schaffalitzky and Zisserman 2002) or location recognition (Chum *et al.* 2007), full affine invariance is preferred. Affine invariant detectors not only respond at consistent locations after scale and orientation changes, they also respond consistently across affine deformations such as (local) perspective foreshortening (Figure 4.13). In fact, for a small enough patch, any continuous image warping can be well approximated by an affine deformation.

To introduce affine invariance, several authors have proposed fitting an ellipse to the auto-correlation or Hessian matrix (using eigenvalue analysis) and then using the principal axes and ratios of this fit as the affine coordinate frame (Lindeberg and Gørding 1997, Baumberg 2000, Mikolajczyk and Schmid 2004, Mikolajczyk *et al.* 2005, Tuytelaars and Mikolajczyk 2007). Figure 4.14 shows how the square root of the moment matrix can be used to transform local patches into a frame which is similar up to rotation. [Note: May need to fix up the above description.]

Another important affine invariant region detector is the Maximally Stable Extremal Region (MSER) detector developed by Matas *et al.* (2004). To detect MSERs, binary regions are com-

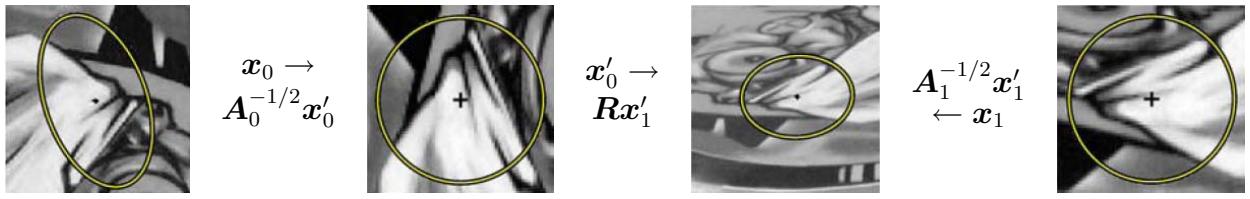


Figure 4.14: *Affine normalization using the second moment matrices, as described in (Mikolajczyk and Schmid 2004)*. After image coordinates are transformed using the matrices $A_0^{-1/2}$ and $A_1^{-1/2}$, they are related by a pure rotation R , which can be estimated using a dominant orientation technique.



Figure 4.15: *Maximally Stable Extremal Regions (MSERs) extracted and matched from a number of images (Matas et al. 2004)*.

puted by thresholding the image at all possible gray levels (the technique therefore only works for grayscale images). This can be performed efficiently by first sorting all pixels by gray value, and then incrementally adding pixels to each connected component as the threshold is changed (Nistér and Stewénius 2008). The area of each component (region) is monitored as the threshold is changed. Regions whose rate of change of area w.r.t. threshold is minimal are defined as *maximally stable* and are returned as detected regions. This results in regions that are invariant to both affine geometric and photometric (linear bias-gain or smooth monotonic) transformations (Figure 4.15). If desired, an affine coordinate frame can be fit to each detected region using its moment matrix.

The area of feature point detectors and descriptors continues to be very active, with papers appearing every year at major computer vision conferences (Xiao and Shah 2003, Koethe 2003, Carneiro and Jepson 2005, Kenney *et al.* 2005, Bay *et al.* 2006, Platel *et al.* 2006, Rosten and Drummond 2006). Mikolajczyk *et al.* (2005) survey a number of popular affine region detectors and provide experimental comparisons of their invariance to common image transformations such as scaling, rotations, noise, and blur. These experimental results, code, and pointers to the surveyed papers can be found on their Web site at <http://www.robots.ox.ac.uk/vgg/research/affine>.

[Note: Clean up the rest of this section. Can I get some suggestions from reviewers as to which are most important?] Of course, keypoints are not the only kind of features that can be used for registering images. Zoghliami *et al.* (1997) use line segments as well as point-like features to estimate homographies between pairs of images, whereas (Bartoli *et al.* 2004) use line segments

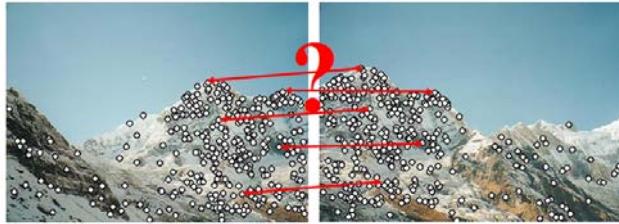


Figure 4.16: *Feature matching: how can we extract local descriptors that are invariant to inter-image variations and yet still discriminative enough to establish correct correspondences?*

with local correspondences along the edges to extract 3D structure and motion. [Note: Check out ([Schmid and Zisserman 1997](#)) and see what they do.] Tuytelaars and Van Gool (2004) use affine invariant regions to detect correspondences for wide baseline stereo matching, whereas Kadir *et al.* (2004) detect salient regions where patch entropy and its rate of change with scale are locally maximal. ([Corso and Hager \(2005\)](#) use a related technique to fit 2D oriented Gaussian kernels to homogeneous regions.) [Note: The following text is from Matas' BMVC 2002 paper: Since the influential paper by Schmid and Mohr [11] many image matching and wide-baseline stereo algorithms have been proposed, most commonly using Harris interest points as distinguished regions. Tell and Carlsson [13] proposed a method where line segments connecting Harris interest points form measurement regions. The measurements are characterized by scale invariant Fourier coefficients. The Harris interest detector is stable over a range of scales, but defines no scale or affine invariant measurement region. Baumberg [1] applied an iterative scheme originally proposed by Lindeberg and Garding to associate affine-invariant measurement regions with Harris interest points. In [7], Mikolajczyk and Schmid show that a scale-invariant MR can be found around Harris interest points. In [9], Pritchett and Zisserman form groups of line segments and estimate local homographies using parallelograms as measurement regions. Tuytelaars and Van Gool introduced two new classes of affine-invariant distinguished regions, one based on local intensity extrema [16] the other using point and curve features [15]. In the latter approach, DRs are characterized by measurements from inside an ellipse, constructed in an affine invariant manner. Lowe [6] describes the “Scale Invariant Feature Transform” approach which produces a scale and orientation-invariant characterization of interest points.]

More details on techniques for finding and matching curves, lines, and regions, can be found in subsequent sections of this chapter.

4.1.2 Feature descriptors

After detecting the features (keypoints), we must *match* them, i.e., determine which features come from corresponding locations in different images. In some situations, e.g., for video sequences ([Shi and Tomasi 1994](#)) or for stereo pairs that have been *rectified* ([Zhang *et al.* 1995](#), Loop and

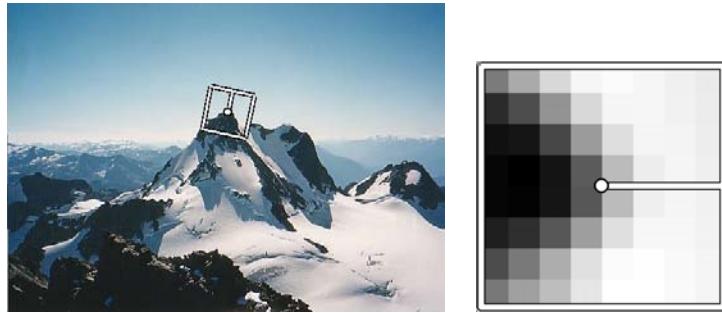


Figure 4.17: *MOPS descriptors are formed using an 8×8 sampling of bias/gain normalized intensity values, with a sample spacing of 5 pixels relative to the detection scale (Brown et al. 2005). This low frequency sampling gives the features some robustness to interest point location error, and is achieved by sampling at a higher pyramid level than the detection scale.*

Zhang 1999, Scharstein and Szeliski 2002), the local motion around each feature point may be mostly translational. In this case, the simple error metrics such as the *sum of squared differences* or *normalized cross-correlation*, described in §7.1, can be used to directly compare the intensities in small patches around each feature point. (The comparative study by Mikolajczyk and Schmid (2005) discussed below uses cross-correlation.) Because feature points may not be exactly located, a more accurate matching score can be computed by performing incremental motion refinement as described in §7.1.3, but this can be time consuming and can sometimes even decrease performance (Brown et al. 2005).

In most cases, however, the local appearance of features will change in orientation, scale, and even affine frame between images. Extracting a local scale, orientation, and/or affine frame estimate and then using this to resample the patch before forming the feature descriptor is thus usually preferable (Figure 4.17).

Even after compensating for these changes, the local appearance of image patches will usually still vary from image to image. How can we make the descriptor that we match more invariant to such changes, while still preserving discriminability between different (non-corresponding) patches (Figure 4.16)? Mikolajczyk and Schmid (2005) review some recently developed view-invariant local image descriptors and experimentally compare their performance. Below, I describe a few of these descriptor in more detail.

Bias and gain normalization (MOPS). For tasks that do not exhibit large amounts of foreshortening, such as image stitching, simple normalized intensity patches perform reasonably well and are simple to implement (Brown et al. 2005) (Figure 4.17). In order to compensate for slight inaccuracies in the feature point detector (location, orientation, and scale), these Multi-Scale Oriented Patches (MOPS) are sampled at spacing of 5 pixels relative to the detection scale (using a coarser

level of the image pyramid to avoid aliasing). To compensate for affine photometric variations (linear exposure changes, *aka* bias and gain, (3.3)), patch intensities are re-scaled so that their mean is zero and their variance is one.

Scale Invariant Feature Transform (SIFT). SIFT features are formed by computing the gradient at each pixel in a 16×16 window around the detected keypoint, using the appropriate level of the Gaussian pyramid at which the keypoint was detected. The gradient magnitudes are down-weighted by a Gaussian fall-off function (shown as a blue circle), in order to reduce the influence of gradients far from the center, as these are more affected by small misregistrations (Figure 4.18).

In each 4×4 quadrant, a gradient orientation histogram is formed by (conceptually) adding the weighted gradient value to one of 8 orientation histogram bins. To reduce the effects of location and dominant orientation misestimation, each of the original 256 weighted gradient magnitudes is softly added to $2 \times 2 \times 2$ histogram bins using trilinear interpolation. (Softly distributing values to adjacent histogram bins is generally a good idea in any application where histograms are being computed, e.g., for Hough transforms §4.3.2 or local histogram equalization §3.1.4.)

The resulting 128 non-negative values form a raw version of the SIFT descriptor vector. To reduce the effects of contrast/gain (additive variations are already removed by the gradient), the 128-D vector is normalized to unit length. To further make the descriptor robust to other photometric variations, values are clipped to 0.2 and the resulting vector is once again renormalized to unit length.

PCA-SIFT Ke and Sukthankar (2004) propose a (simpler to compute) descriptor inspired by SIFT, which computes the x and y (gradient) derivatives over a 39×39 patch and then reduces the resulting 3042-dimensional vector to 36 using PCA (Principal Component Analysis). [*Note: Where do I introduce PCA in the book?*]

Gradient location-orientation histogram (GLOH) This descriptor, developed by Mikolajczyk and Schmid (2005), is a variant on SIFT that uses a log-polar binning structure instead of the 4 quadrants used by Lowe (2004) (Figure 4.19). The spatial bins are of radius 6, 11, and 15, with eight angular bins (except for the central region), for a total of 17 spatial bins and 16 orientation bins. The 272-dimensional histogram is then projected onto a 128 dimensional descriptor using PCA trained on a large database. In their evaluation, Mikolajczyk and Schmid (2005) found that GLOH, which has the best performance overall, outperforms SIFT by a small margin.

Steerable filters Steerable filters, §3.2.1, are combinations of derivative of Gaussian filters that permit the rapid computation of even and odd (symmetric and anti-symmetric) edge-like and corner-like features at all possible orientations (Freeman and Adelson 1991). Because they use

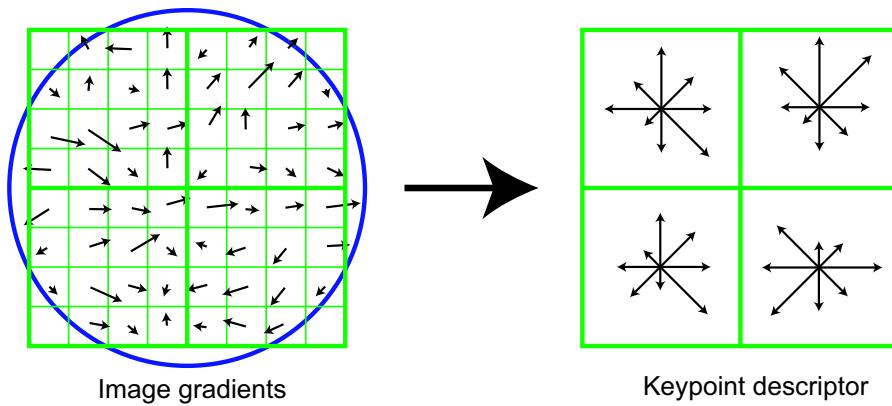


Figure 4.18: A schematic representation of Lowe's (2004) Scale Invariant Feature Transform (SIFT). Gradient orientations and magnitudes are computed at each pixel and then weighted by a Gaussian falloff (blue circle). A weighted gradient orientation histogram is then computed in each subregion, using trilinear interpolation. While this figure shows an 8×8 pixel patch and a 2×2 descriptor array, Lowe's actual implementation uses 16×16 patches and a 4×4 array of 8-bin histograms.

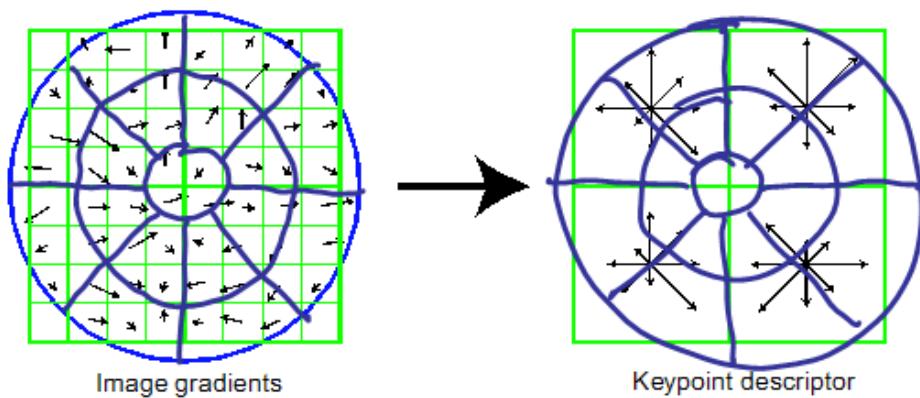


Figure 4.19: The Gradient Location-Orientation Histogram (GLOH) descriptor uses log-polar bins instead of square bins to compute orientation histograms (Mikolajczyk and Schmid 2005). [Note: This diagram is not yet to scale, i.e., the radii should be larger, and there should be a corresponding RHS to the figure. Redraw it properly in Visio when you have more time.]

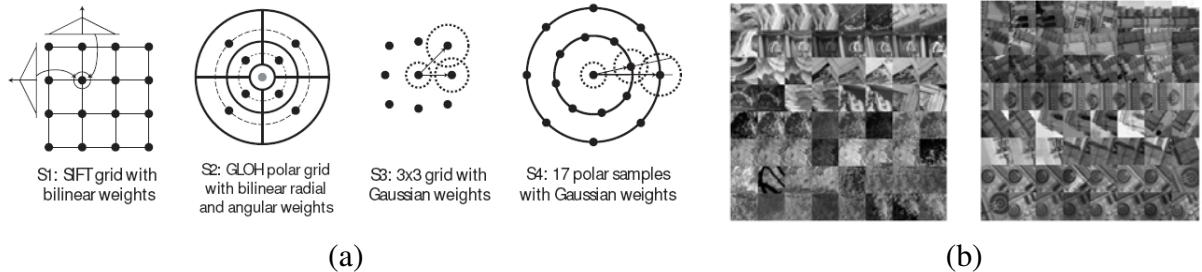


Figure 4.20: Spatial summation blocks for SIFT, GLOH, and some newly developed feature descriptors (Winder and Brown 2007). The parameters for the new features (a), e.g., their Gaussian weights, are learned from a training database of matched real-world image patches (b) obtained from robust structure-from-motion applied to Internet photo collections (Hua et al. 2007, Snavely et al. 2006).

reasonably broad Gaussians, they too are somewhat insensitive to localization and orientation errors.

Performance of local descriptors Among the local descriptors that Mikolajczyk and Schmid (2005) compared, they found that GLOH performed the best, followed closely by SIFT (Figure 4.25). Results for many other descriptors, not covered in this book, are also presented.

The field of feature descriptors continues to evolve rapidly, with some of the newer techniques looking at local color information (van de Weijer and Schmid 2006, Abdel-Hakim and Farag 2006). Winder and Brown (2007) develop a multi-stage framework for feature descriptor computation that subsumes both SIFT and GLOH (Figure 4.20a) and also allows them to learn optimal parameters for newer descriptors that outperform previous hand-tuned descriptors. Hua *et al.* (2007) extend this work by learning lower-dimensional projections of higher-dimensional descriptors that have the best discriminative power. Both of these papers use a database of real-world image patches (Figure 4.20b) obtained by sampling images at locations that were reliably matched using a robust structure-from-motion algorithm applied to Internet photo collections (Snavely *et al.* 2006, Goesele *et al.* 2007).

While these techniques construct feature detectors that optimize for repeatability across *all* object classes, it is also possible to develop class- or instance-specific feature detectors that maximize *discriminability* from other classes (Ferencz *et al.* 2008).

4.1.3 Feature matching

Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images. The approach we take depends



Figure 4.21: *Recognizing objects in a cluttered scene (Lowe 2004)*. Two of the training images in the database are shown on the left. These are matched to the cluttered scene in the middle using SIFT features, shown as small squares in the right image. The affine warping of each recognized database image onto the scene is shown as a larger parallelogram in the left image.

partially on the application, e.g., different strategies may be preferable for matching images that are known to overlap (e.g., in image stitching) vs. images that may have no correspondence whatsoever (e.g., when trying to recognize objects from a database).

In this section, I divide this problem into two separate components. The first is to select a *matching strategy*, which determines which correspondences are passed on to the next stage for further processing. The second is to devise efficient *data structures* and *algorithms* to perform this matching as quickly as possible. (See the discussion of related techniques in the chapter on recognition §14.3.2.)

Matching strategy and error rates

As I mentioned before, the determining which features matches are reasonable to further process depends on the context in which the matching is being performed. Say we are given two images that overlap to a fair amount (e.g., for image stitching, as in Figure 4.16, or for tracking objects in a video). We know that most features in one image are likely to match the other image, although some may not match because they are occluded or their appearance has changed too much.

On the other hand, if we are trying to recognize how many known objects appear in a cluttered scene (Figure 4.21), most of the features may not match. Furthermore, a large number of potentially matching objects must be searched, which requires more efficient strategies, as described below.

To begin with, we assume that the feature descriptors have been designed so that Euclidean (vector magnitude) distances in feature space can be used for ranking potential matches. If it

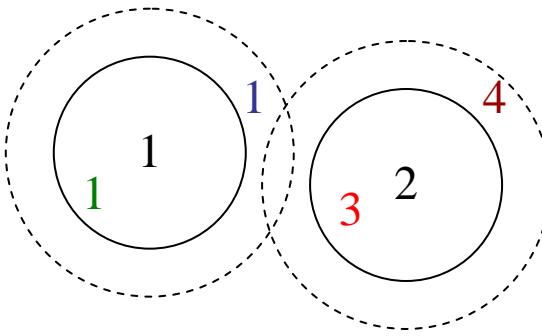


Figure 4.22: An example of false positives and negatives. The black digits 1 and 2 are the features being matched against a database of features in other images. At the current threshold setting (black circles), the green 1 is a true positive (good match), the blue 1 is a false negative (failure to match), and the red 3 is a false positive (incorrect match). If we set the threshold higher (dashed circle), the blue 1 becomes a true positive, but the brown 4 becomes an additional false positive.

turns out that certain parameters (axes) in a descriptor are more reliable than others, it is usually preferable to re-scale these axes ahead of time, e.g., by determining how much they vary when compared against other known good matches (Hua *et al.* 2007).

Given a Euclidean distance metric, the simplest matching strategy is to set a threshold (maximum distance) and to return all matches from other images within this threshold. Setting the threshold too high results in too many *false positives*, i.e., incorrect matches being returned. Setting the threshold too low results in too many *false negatives*, i.e., too many correct matches being missed (Figure 4.22).

We can quantify the performance of a matching algorithm at a particular threshold by first counting the number of true and false matches and match failures, using the following definitions (Fawcett 2006):

- **TP:** true positives, i.e., number of correct matches;
- **FN:** false negatives, matches that were not correctly detected;
- **FP:** false positives, estimated matches that are incorrect;
- **TN:** true negatives, non-matches that were correctly rejected.

Table 4.1 shows a sample *confusion matrix* (contingency table) containing such numbers.

We can convert these numbers into *unit rates* by defining the following quantities (Fawcett 2006):

- true positive rate **TPR**,

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}}; \quad (4.14)$$

	True matches	True non-match.	
Pred. matches	TP = 18	FP = 4	P' = 22
Pred. non-match.	FN = 2	TN = 76	N' = 78
	P = 20	N = 80	Total = 100
	TPR = 0.90	FPR = 0.05	ACC = 0.94

Table 4.1: Sample table showing the number of matches correctly and incorrectly estimated by a feature matching algorithm. The table shows the number true positives (TP), false negatives (FN), false positives (FP), true negatives (TN). The columns sum up to the actual number of positives (P) and negatives (N), while the rows sum up to the estimated number of positives (P') and negatives (N'). The formulas for the true positive rate (TPR), the false positive rate (FPR), the positive predictive value (PPR), and the accuracy (ACC) are given in the text.

- false positive rate FPR,

$$FPR = \frac{FP}{FP+TN} = \frac{FP}{N}; \quad (4.15)$$

- positive predictive value (PPR),

$$PPR = \frac{TP}{TP+TN} = \frac{TP}{P'}; \quad (4.16)$$

- accuracy (ACC),

$$ACC = \frac{TP+TN}{P+N}. \quad (4.17)$$

Again, table 4.1 shows some sample numbers. [Note: See if any of these are in (Hastie et al. 2001, Bishop 2006).]

In the *information retrieval* (or document retrieval) literature, the terms *precision* is used instead of PPV (how many returned documents are relevant) and *recall* is used instead of TPR (what fraction of relevant documents was found).

Any particular matching strategy (at a particular threshold or parameter setting) can be rated by the TPR and FPR numbers: ideally, the true positive rate will be close to 1, and the false positive rate close to 0. As we vary the matching threshold, we obtain a family of such points, which are collectively known as the *Receiver Operating Characteristic* or *ROC curve* (Fawcett 2006) (Figure 4.23a). The closer this curve lies to the upper left corner, i.e., the larger the area under the curve (AUC), the better its performance. Figure 4.23b shows how we can plot the number of matches and non-matches as a function of inter-feature distance d . These curves can then be used to plot an ROC curve (Exercise 4.4).

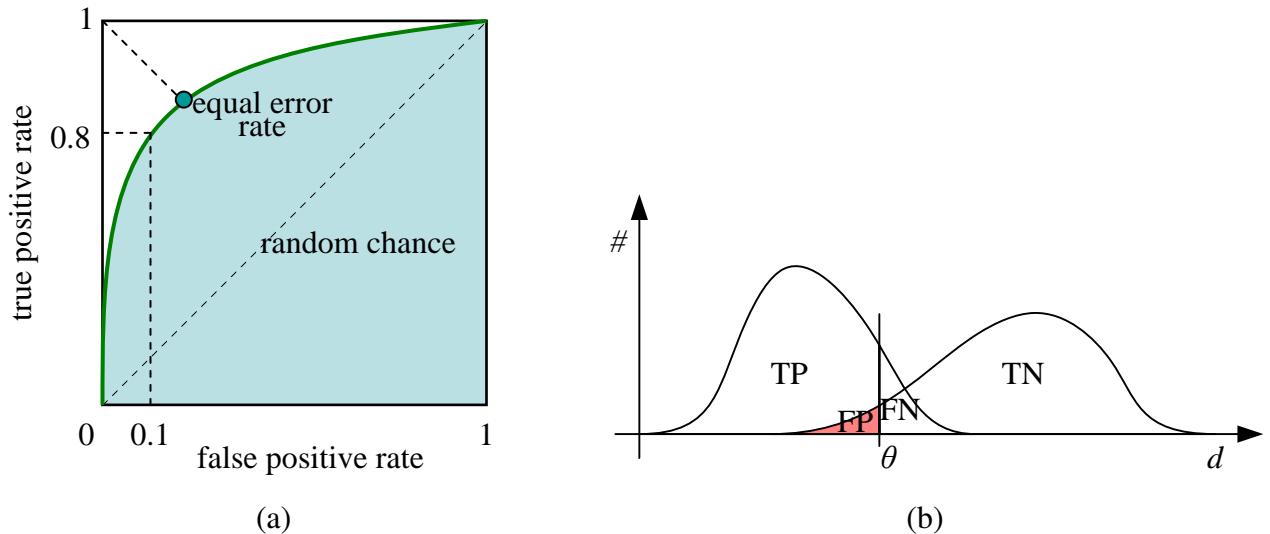


Figure 4.23: *ROC curve and its related rates.* (a) The ROC curve plots the true positive rate against the false negative rate for a particular combination of feature extraction and matching algorithms. Ideally, the true positive rate should be close to 1, while the true negative rate is close to 0. The area under the ROC curve (AUC) is often used as a single (scalar) measure of algorithm performance. Alternatively, the equal error rate is sometimes used. (b) The distribution of positives (matches) and negatives (non-matches) as a function of inter-feature distance d . As the threshold θ is increased, the number of true positives (TP) and false positives (FP) increases.

The problem with using a fixed threshold is that it is difficult to set; the useful range of thresholds can vary a lot as we move to different parts of the feature space (Lowe 2004, Mikolajczyk and Schmid 2005). A better strategy in such cases is to simply match the *nearest neighbor* in feature space. Since some features may have no matches (e.g., they may be part of background clutter in object recognition, or they may be occluded in the other image), a threshold is still used to reduce the number of false positives.

Ideally, this threshold itself will adapt to different regions of the feature space. If sufficient training data is available (Hua *et al.* 2007), it is sometimes possible to learn different thresholds for different features. Often, however, we are simply given a collection of images to match, e.g., when stitching images or constructing 3D models from unordered photo collections (Brown and Lowe 2003a, Brown and Lowe 2003b, Snavely *et al.* 2006). In this case, a useful heuristic can be to compare the nearest neighbor distance to that of the second nearest neighbor, preferably taken from an image that is known not to match the target (e.g., a different object in the database) (Brown and Lowe 2002, Lowe 2004). We can define this *nearest neighbor distance ratio* (Mikolajczyk and

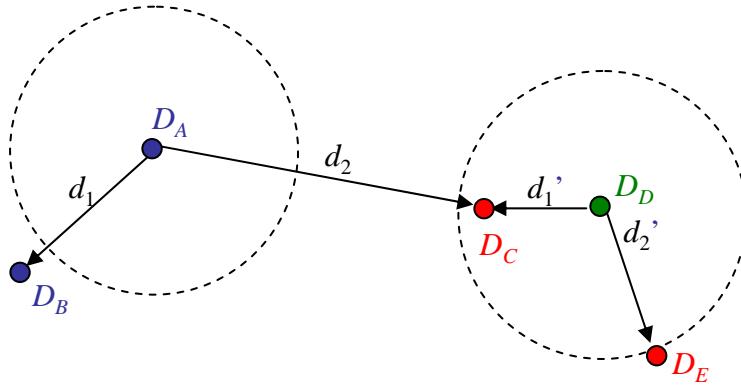


Figure 4.24: *Fixed threshold, nearest neighbor, and nearest neighbor distance ratio matching.* At a fixed distance threshold (dashed circles), descriptor D_A fails to match D_B , and D_D incorrectly matches D_C and D_E . If we pick the nearest neighbor, D_A correctly matches D_B , but D_D incorrectly matches D_C . Using nearest neighbor distance ratio (NNDR) matching, the small NNDR d_1/d_2 correctly matches D_A with D_B , and the large NNDR d'_1/d'_2 correctly rejects matches for D_D .

Schmid 2005) as

$$\text{NNDR} = \frac{d_1}{d_2} = \frac{\|D_A - D_B\|}{\|D_A - D_C\|}, \quad (4.18)$$

where d_1 and d_2 are the nearest and second nearest neighbor distances, and D_A, \dots, D_C are the target descriptor along with its closest two neighbors (Figure 4.24).

The effects of using these three different matching strategies for the feature descriptors evaluated by Mikolajczyk and Schmid (2005) can be seen in Figure 4.25. As you can see, the nearest neighbor and NNDR strategies produce improved ROC curves.

Efficient matching

Once we have decided on a matching strategy, we still need to efficiently search for potential candidates. The simplest way to find all corresponding feature points is to compare all features against all other features in each pair of potentially matching images. Unfortunately, this is quadratic in the number of extracted features, which makes it impractical for most applications.

A better approach is to devise an *indexing structure* such as a multi-dimensional search tree or a hash table to rapidly search for features near a given feature. Such indexing structures can either be built for each image independently (which is useful if we want to only consider certain potential matches, e.g., searching for a particular object), or globally for all the images in a given database, which can potentially be faster, since it removes the need to iterate over each image. For extremely large databases (millions of images or more), even more efficient structures based on ideas from

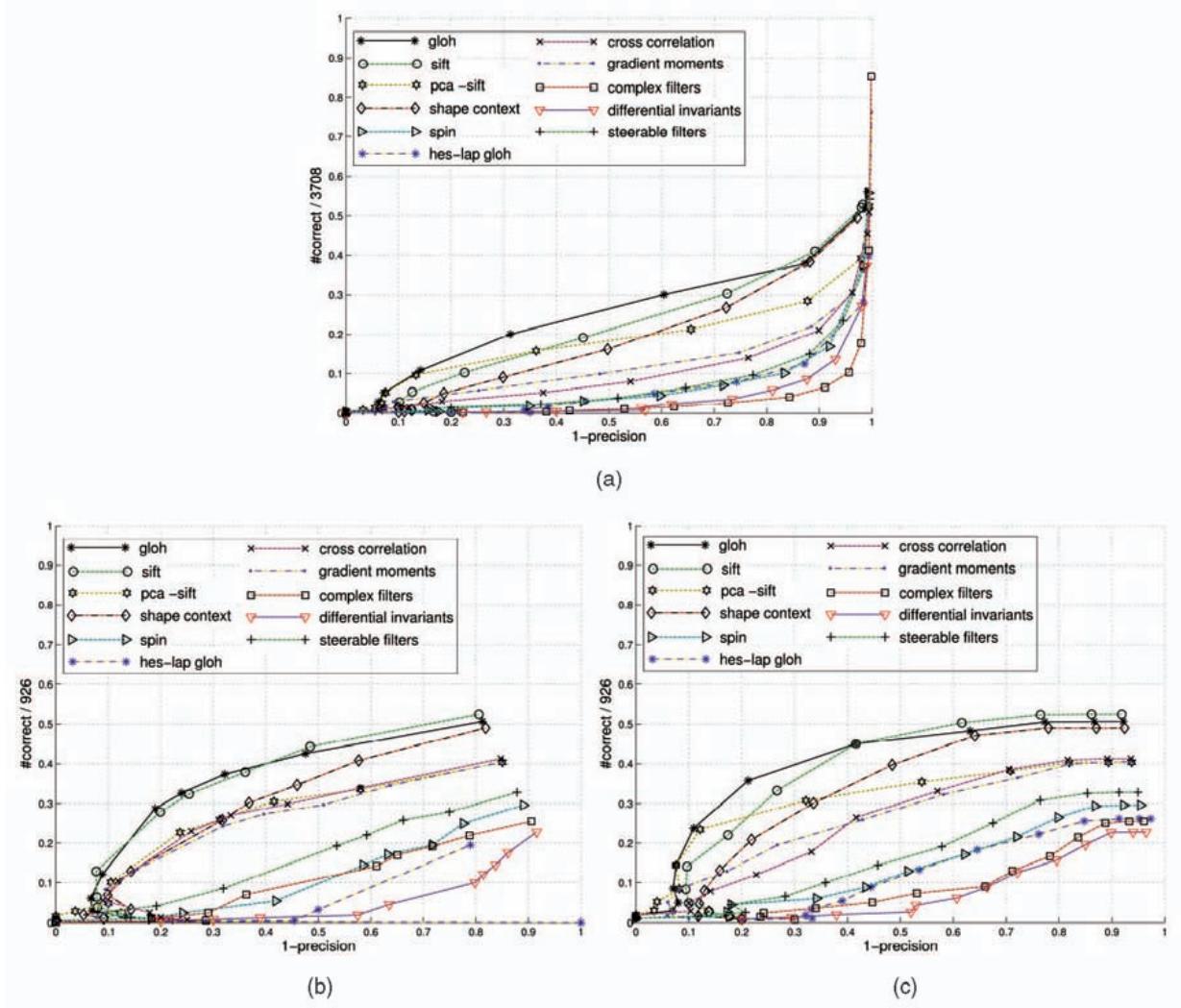


Figure 4.25: *Performance of the feature descriptors evaluated by Mikolajczyk and Schmid (2005), shown for three different matching strategies: (a) fixed threshold; (b) nearest neighbor; (c) nearest neighbor distance ratio (NNDR). Note how the ordering of the algorithms does not change that much, but the overall performance varies significantly between ...*

[Note: Need to check what the definition of precision is...]

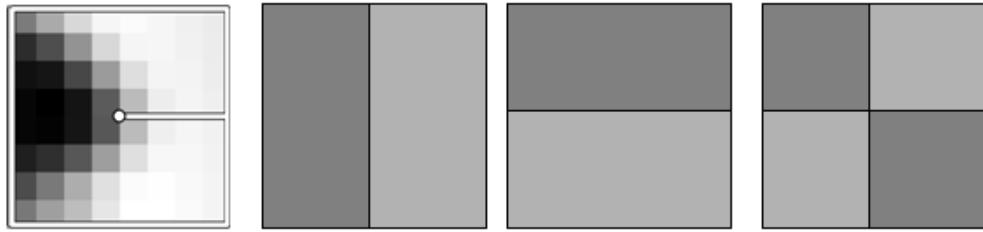


Figure 4.26: The three Haar wavelet coefficients used for hashing the MOPS descriptor devised by Brown *et al.* (2005) are computed by summing each 8×8 normalized patch over the light and dark gray regions and taking their difference.

document retrieval (e.g., *vocabulary trees*, (Nister and Stewenius 2006)) can be used §14.3.2.

One of the simpler techniques to implement is multi-dimensional hashing, which maps descriptors into fixed size buckets based on some function applied to each descriptor vector. At matching time, each new feature is hashed into a bucket, and a search of nearby buckets is used to return potential candidates (which can then be sorted or graded to determine which are valid matches).

A simple example of hashing is the Haar wavelets used by Brown *et al.* (2005) in their MOPS paper. During the matching structure construction, each 8×8 scaled, oriented, and normalized MOPS patch is converted into a 3-element index by performing sums over different quadrants of the patch (Figure 4.26). The resulting three values are normalized by their expected standard deviations and then mapped to the two (of $b = 10$) nearest 1-D bins. The three-dimensional indices formed by concatenating the three quantized values are used to index the $2^3 = 8$ bins where the feature is stored (added). At query time, only the primary (closest) indices are used, so only a single three-dimensional bin needs to be examined. The coefficients in the bin can then be used to select k approximate nearest neighbors for further processing (such as computing the NNDR).

A more complex, but more widely applicable, version of hashing is called *locality-sensitive hashing*, which uses unions of independently computed hashing functions to index the features (Gionis *et al.* 1999, Shakhnarovich *et al.* 2006). Shakhnarovich *et al.* (2003) extend this technique to be more sensitive to the distribution of points in parameter space, which they call *parametersensitive hashing*. [Note: Should I say more about these? Read the papers and then decide.]

Another widely used class of indexing structures are multi-dimensional search trees. The best known of these are called *k-D trees*, which divide the multi-dimensional feature space along alternating axis-aligned hyperplanes, choosing the threshold along each axis so as to maximize some criterion such as the search tree balance (Samet 1989). Figure 4.27 shows an example of a two-dimensional *k-d* tree. Here, eight different data points A–H are shown as small diamonds arranged on a two-dimensional plane. The *k-d* tree recursively splits this plane along axis-aligned (hor-

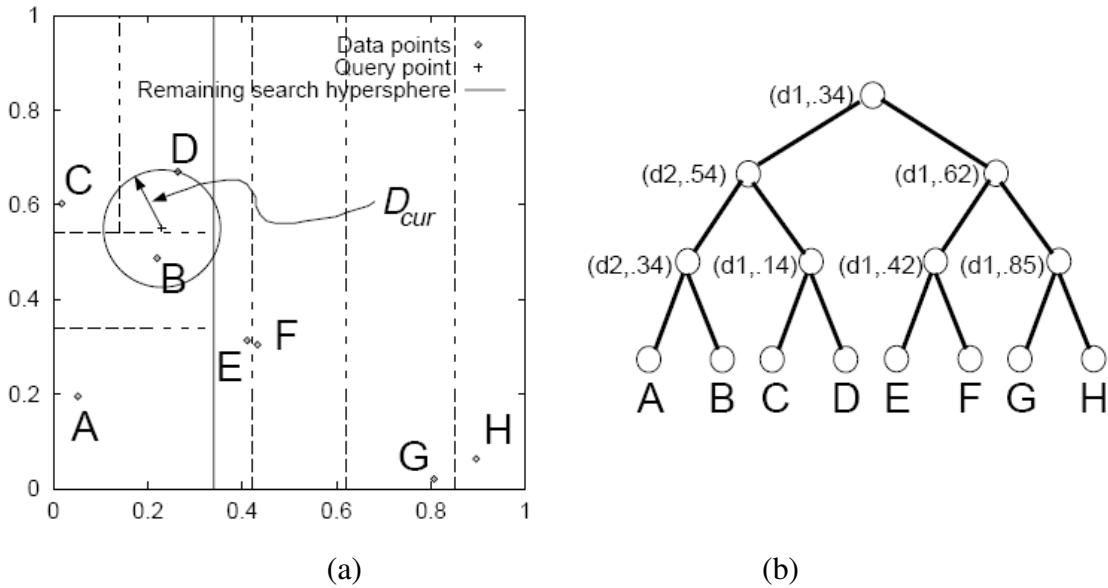


Figure 4.27: *k*-D tree and Best Bin First (BBF), from (Beis and Lowe 1999). (a) The spatial arrangement of the axis-aligned cutting planes is shown using dashed lines. Individual data points are shown as small diamonds. (b) The same subdivision can be represented as a tree, where each interior node represents an axis-aligned cutting plane (e.g., the top node cuts along dimension d_1 at value .34), and each leaf node is a data point. During Best Bin First (BBF) search, a query point, denoted by a '+', first looks in its containing bin (D) and then in its nearest adjacent bin (B), rather than its closest neighbor in the tree (C).

zontal or vertical) cutting planes. Each split can be denoted using the dimension number and split value (Figure 4.27b). The splits are arranged so as to try to balance the tree (keep its maximum depths as small as possible). At query time, a classic k -d tree search first locates the query point (+) in its appropriate bin (D), and then searches nearby leaves in the tree (C, B, ...) until it can guarantee that the nearest neighbor has been found. The Best Bin First (BBF) search (Beis and Lowe 1999) searches bins in order of their spatial proximity to the query point, and is therefore usually more efficient.

Many additional data structures have been developed over the years for solving nearest neighbor problems (Arya *et al.* 1998, Liang *et al.* 2001, Hjaltason and Samet 2003). For example, Nene and Nayar (1997) developed a technique they call *slicing* that uses a series of 1D binary searches on the point list sorted along different dimensions to efficiently cull down a list of candidate points that lie within a hypercube of the query point. Grauman and Darrell (2005) re-weight the matches at different levels of an indexing tree, which allows their technique to be less sensitive to discretization errors the tree construction. [Note: Is this right? Need to re-read.] Even more recently,

Nister and Stewenius (2006) use a *metric tree*, which consists of comparing feature descriptors to a small number of prototypes at each level in a hierarchy. The resulting quantized *visual words* can then be used with classical information retrieval (document relevance) techniques to quickly winnow down a set of potential candidates from a database of millions of images. Despite all of this promising work, the rapid computation of image feature correspondences remains a challenging open research problem.

Feature match verification and densification

Once we have some hypothetical (putative) matches, we can often use geometric alignment (§5.1) to verify which matches are *inliers* and which ones are *outliers*. For example, if we expect the whole image to be translated or rotated in the matching view, we can fit a global geometric transform and keep only those feature matches that are sufficiently close to this estimated transformation. The process of selecting a small set of seed matches and then verifying a larger set is often called *random sampling* or RANSAC (§5.1.5). Once an initial set of correspondences has been established, some systems look for additional matches, e.g., by looking for additional correspondences along epipolar lines (§10.1) or in the vicinity of estimated locations based on the global transform. These topics will be discussed further in Sections §5.1 and §10.2.

4.1.4 Feature tracking

An alternative to independently finding features in all candidate images and then matching them is to find a set of likely feature locations in a first image and to then *search* for their corresponding locations in subsequent images. This kind of *detect then track* approach is more widely used for video tracking applications, where the expected amount of motion and appearance deformation between adjacent frames is expected to be small (or at least bounded).

The process of selecting good features to track is closely related to selecting good features for more general recognition applications. In practice, regions containing high gradients in both directions, i.e., which have high eigenvalues in the auto-correlation matrix (4.8), provide stable locations at which to find correspondences (Shi and Tomasi 1994).

In subsequent frames, searching for locations where the corresponding patch has low squared difference (4.1) often works well enough. However, if the images are undergoing brightness change, explicitly compensating for such variations (7.9) or using *normalized cross-correlation* (7.11) may be preferable. If the search range is large, it is also often more efficient to use a *hierarchical* search strategy, which uses matches in lower-resolution images to provide better initial guesses and hence speed up the search (§7.1.1). These topics are all covered in more detail in the section on motion estimation (§7.1).

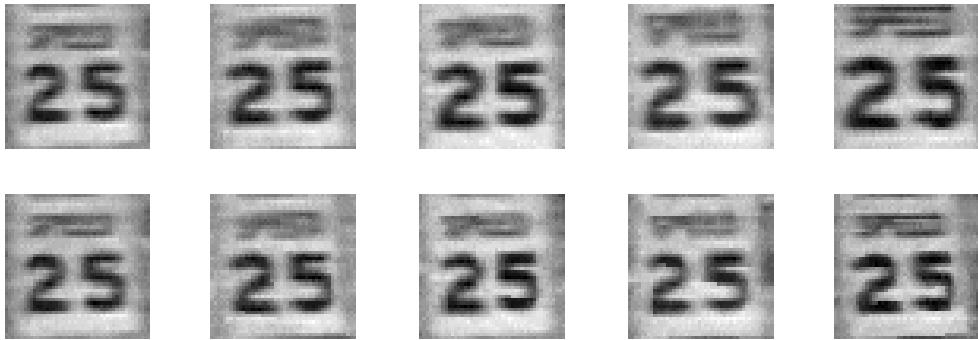


Figure 4.28: *Feature tracking using an affine motion model (Shi and Tomasi 1994).* Top row: image patch around the tracked feature location. Bottom row: image patch after warping back toward the first frame using an affine deformation. Even though the speed sign gets larger from frame to frame, the affine transformation maintains a good resemblance between the original and subsequent tracked frames.

If features are being tracked over longer image sequences, their appearance can undergo larger changes. You then have to decide whether to continue matching against the originally detected patch (feature), or to re-sample each subsequent frame at the matching location. The former strategy is prone to failure as the original patch can undergo appearance changes such as foreshortening. The latter runs the risk of the feature drifting from its original location to some other location in the image (Shi and Tomasi 1994). (Mathematically, one can argue that small mis-registration errors compound to create a *Markov Random Walk*, which leads to larger drift over time.)

A preferable solution is to compare the original patch to later image locations using an *affine* motion model (§7.2). Shi and Tomasi (1994) first compare patches using a translational model between neighboring frames, and then use the location estimate produced by this step to initialize an affine registration between the patch in the current frame and the base frame where a feature was first detected (Figure 4.28). In their system, features are only detected infrequently, i.e., only in regions where tracking has failed. In the usual case, an area around the current *predicted* location of the feature is searched with an incremental registration algorithm (§7.1.3).

Since their original work on feature tracking, Shi and Tomasi's approach has generated a string of interesting follow-on papers and applications. Beardsley *et al.* (1996) use extended feature tracking combined with structure from motion (§6) to incrementally build up sparse 3-D models from video sequences. Kang *et al.* (1997) tie together the corners of adjacent (regularly gridded) patches to provide some additional stability to the tracking (at the cost of poorer handling of occlusions). Tommasini *et al.* (1998) provide a better spurious match rejection criterion for the basic Shi and Tomasi algorithm, Collins and Liu (2003) provide improved mechanisms for feature selection and dealing with larger appearance changes over time, and Shafique and Shah (2005) develop al-



Figure 4.29: *Real-time head tracking using the fast trained classifiers of Lepetit et al. (2005).*

gorithms for feature matching (data association) for videos with large numbers of moving objects or points.

One of the newest developments in feature tracking is the use of learning algorithms to build special-purpose recognizers to rapidly search for matching features anywhere in an image (Lepetit *et al.* 2005, Hinterstoisser *et al.* 2008, Rogez *et al.* 2008). By taking the time to train classifiers on sample patches and their affine deformations, extremely fast and reliable feature detectors can be constructed, which enables much faster motions to be supported. (Figure 4.29). Coupling such features to deformable models results in even higher stability (Pilet *et al.* 2008).

4.1.5 Application: Performance-driven animation

One of the most compelling applications of fast feature tracking is *performance-driven animation*, i.e., the interactive deformation of a 3D graphics model based on tracking a user’s motions (Williams 1990, Litwinowicz and Williams 1994, Lepetit *et al.* 2005).

Buck *et al.* (2000) present a system for tracking a user’s facial expressions and head motion and using these to morph among a series of hand-drawn sketches. The system starts by extracting the eye and mouth regions of each sketch and drawing control lines over each image (Figure 4.30a).

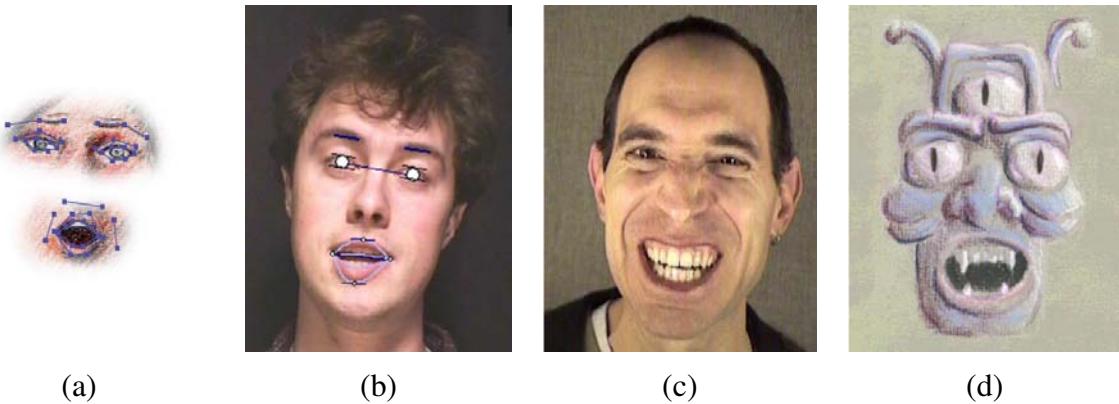


Figure 4.30: *Performance-driven hand-drawn animation* (Buck et al. 2000). (a) eye and mouth portions of hand-drawn sketch with their overlaid control lines; (b) an input video frame with the tracked features overlaid; (c) a different input video frame along with its (d) corresponding hand-drawn animation.

At run time, a face tracking system (see (Toyama 1998) for a survey of such systems) determines the current location of these features (Figure 4.30b). The animation system decides which input images to morph based on nearest neighbor feature appearance matching and triangular barycentric interpolation. It also computes The global location and orientation of the head from the tracked features. The resulting morphed eye and mouth regions are then composited back into the overall head model to yield a frame of hand-drawn animation (Figure 4.30d).

4.2 Edges

While interest points are useful for finding image locations that can be accurately matched in 2D, edge points are far more plentiful and often carry important semantic associations. For example, the boundaries of objects, which also correspond to occlusion events in 3D, are usually delineated by visible contours. Other kinds of edges correspond to shadow boundaries or crease edges, where surface orientation changes rapidly. “Line drawings”, which consist solely of drawn contours, are popular elements in books for infants, who seem to have no difficulty in recognizing familiar objects or animals from such simple depictions. [Note: Find a reference from the visual perception / psychophysics literature?] Isolated edge points can also be grouped longer curves or contours, as well as straight line segments §4.3.

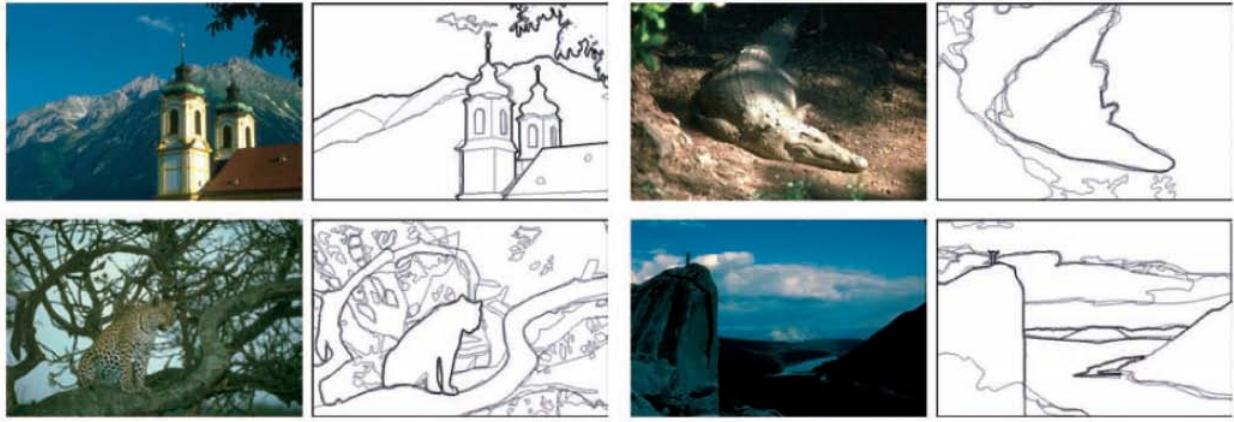


Figure 4.31: *Human boundary detection* (Martin et al. 2004). The darkness of the edges corresponds to how many human subjects marked an object boundary at that location.

4.2.1 Edge detection

Given an image, how can we find the salient edges? Consider the color images in Figure 4.31. If someone asked you to point out the most “salient” or “strongest” edges or the object boundaries (Martin et al. 2004), which ones would you trace? How closely do your perceptions match the edge images shown in Figure 4.31.

Qualitatively, edges occur at boundaries between regions of different color, intensity, or texture. Unfortunately, segmenting an image into coherent regions is a difficult task, which I will address later in §4.5. Often, it is preferable to detect edges using only purely local information.

Under such conditions, a reasonable approach is to define an edge as a location of *rapid intensity variation*.² Think of an image as a height field (Figure 4.32). On such a surface, edges occur at locations of *steep slopes*, or equivalently, in regions of closely packed contour lines (on a topographic map).

A mathematical way to define the slope (and direction) of a surface is through its gradient,

$$\mathbf{J}(\mathbf{x}) = \nabla I(\mathbf{x}) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)(\mathbf{x}). \quad (4.19)$$

The local gradient vector \mathbf{J} points in the direction of *steepest ascent* in the intensity function. Its magnitude is an indication of the slope or strength of the variation, while its orientation points in a direction *perpendicular* to the local contour (Figure 4.32).

Unfortunately, taking image derivatives accentuates high frequencies and hence amplifies noise (since the proportion of noise to signal is larger at high frequencies). It is therefore necessary to smooth the image with a low-pass filter prior to computing the gradient. Because we would like the

² I defer the topic of edge detection in color images to the next sub-section.

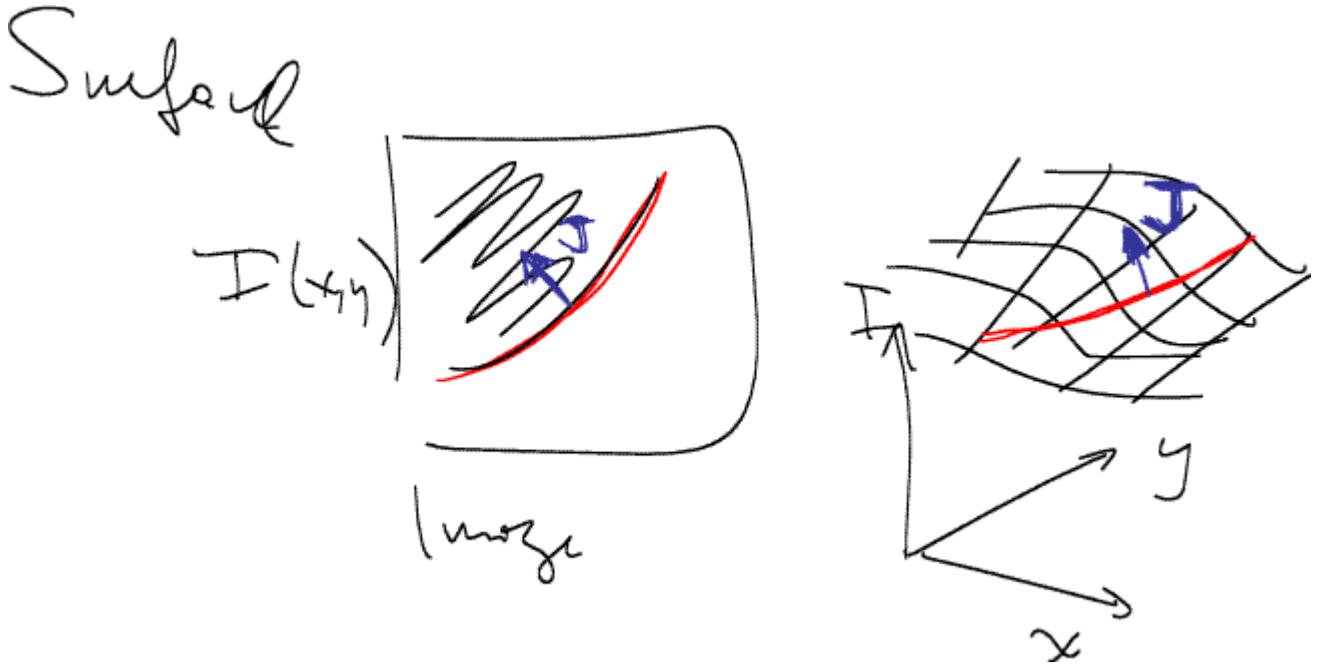


Figure 4.32: *Edge detection as slope finding in a height field. The local gradient vector points in the direction of greatest ascent and is perpendicular to the edge/contour orientation.*

response of our edge detector to be independent of orientation, a circularly symmetric smoothing filter is desirable. As we saw in §3.2.1, the Gaussian is the only separable circularly-symmetric filter, and so it is used in most edge detection algorithms. Canny (1986) discusses alternative filters, and (Nalwa and Binford 1986, Nalwa 1987, Deriche 1987, Freeman and Adelson 1991, Nalwa 1993, Heath *et al.* 1998, Crane 1997, Ritter and Wilson 2000, Bowyer *et al.* 2001) review alternative edge detection algorithms and compare their performance.

Because differentiation is a linear operation, it commutes with other linear filtering operations. The gradient of the smoothed image can therefore be written as

$$\mathbf{J}_\sigma(\mathbf{x}) = \nabla[G_\sigma(\mathbf{x}) * I(\mathbf{x})] == [\nabla G_\sigma](\mathbf{x}) * I(\mathbf{x}), \quad (4.20)$$

i.e., we can convolve the image with the horizontal and vertical derivatives of the Gaussian kernel function,

$$\nabla G_\sigma(\mathbf{x}) = \left(\frac{\partial G_\sigma}{\partial x}, \frac{\partial G_\sigma}{\partial y} \right)(\mathbf{x}) = [-x - y] \frac{1}{\sigma^3} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (4.21)$$

(The parameter σ indicates the width of the Gaussian.) This is the same computation that is performed by Freeman and Adelson's (1991) first order steerable filter, which we already covered in §3.2.1 (3.27–3.28).

Figure 4.33b–c shows the result of convolving an image with the gradient of two different Gaussians. The direction of the gradient is encoded as the hue, and its strength is encoded as the

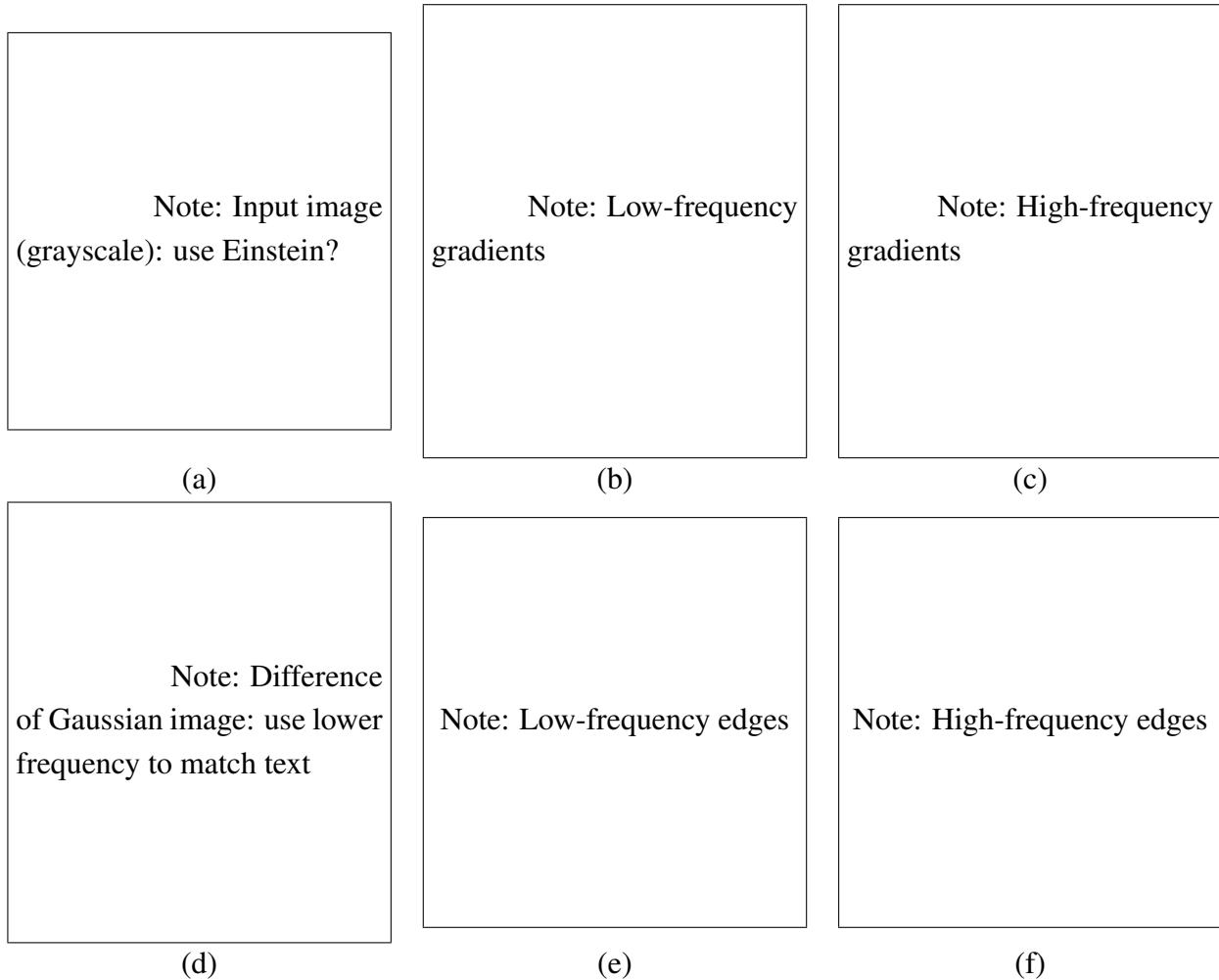


Figure 4.33: *Edge detection: (a) input image; (b) low-frequency gradients, color-coded for orientation and strength; (c) high-frequency gradients; (d) band-pass (Difference of Gaussian) filtered image; (e) low-frequency edges; (f) high-frequency edges. The detected edges are coded both by their strength (saturation) and orientation (hue).* [Note: Generate these images. How do I detect edges? Multiply zero crossing by strength? May also want an additional row of results for just the gradient magnitude?]

saturation/value. As you can see, the gradients are stronger in areas corresponding to what we would perceive as edges.

For many applications, however, we wish to *thin* such a continuous image to only return isolated edges, i.e., as single pixels or *edgels* at discrete locations along the edge contours. This can be achieved by looking for *maxima* in the edge strength (gradient magnitude) in a direction *perpendicular* to the edge orientation, i.e., along the gradient direction. [Note: It may be more convenient to just define the edge orientation as being the same as that of the (signed) gradient, i.e., pointing from dark to light. See Figure 2.2 for an illustration of the normal vector $\hat{\mathbf{n}}$ used to represent a line, and the local line equation becomes $(\mathbf{x} - \mathbf{x}_i) \cdot \hat{\mathbf{n}}_i = 0$. This will make subsequent Hough transform explanation simpler.]

Finding this maximum corresponds to taking a directional derivative of the strength field in the direction of the gradient and then looking for zero crossings. The desired directional derivative is equivalent to the dot product between a second gradient operator and the results of the first,

$$S_\sigma(\mathbf{x}) = \nabla \cdot \mathbf{J}_\sigma(\mathbf{x}) = [\nabla^2 G_\sigma](\mathbf{x}) * I(\mathbf{x}). \quad (4.22)$$

The gradient operator dot product with the gradient is called the *Laplacian*. The convolution kernel

$$\nabla^2 G_\sigma(\mathbf{x}) = \frac{1}{\sigma^3} \left(2 - \frac{x^2 + y^2}{2\sigma^2} \right) \exp \left(-\frac{x^2 + y^2}{2\sigma^2} \right) \quad (4.23)$$

is therefore called the *Laplacian of Gaussian* (LoG) kernel. This kernel can be split into two separable parts,

$$\nabla^2 G_\sigma(\mathbf{x}) = \frac{1}{\sigma^3} \left(1 - \frac{x^2}{2\sigma^2} \right) G_\sigma(x) G_\sigma(y) + \frac{1}{\sigma^3} \left(1 - \frac{y^2}{2\sigma^2} \right) G_\sigma(y) G_\sigma(x). \quad (4.24)$$

In practice, it is quite common to replace the Laplacian of Gaussian convolution with a Difference of Gaussian (DoG) computation, since the kernel shapes are qualitatively similar. This is especially convenient if a “Laplacian pyramid” (§3.4) has already been computed.³

Figure 4.33d shows the input image convolved with a Difference of Gaussian at the lower frequency. Taking the zero crossings of such an image (at the appropriate frequency) and multiplying it with the gradient magnitudes results in the edge images shown in Figure 4.33e–f.

In fact, it is not strictly necessary to take differences between adjacent levels when computing the edge field. [Note: Watch out: if you take a larger area average, smaller bar-like edges will have their threshold shifted up/down. Better check if having a larger parent level actually helps.] Think about what a zero crossing in a “generalized” difference of Gaussians image represents. The finer (smaller kernel) Gaussian is a noise-reduced version of the original image. The coarser

³ Recall that Burt and Adelson’s (1983a) “Laplacian pyramid” actually computed differences of Gaussian-filtered levels.

(larger kernel) Gaussian is an estimate of the average intensity over a larger region. Thus, whenever the DoG image changes sign, this corresponds to the (slightly blurred) image going from relatively darker to relatively lighter, as compared to the average intensity in that neighborhood. [Note: In fact, it may not be necessary to compute the gradient magnitude and direction first, but just to operate on the DoG image directly. Need to check what my current code does, and compare the results, or leave that to an exercise.]

Once we have computed the signed function $S(\mathbf{x})$, we must find its *zero crossings* and convert these into edge elements or *edgels*. Figure 4.34a shows a pixel grid with some sample values of $S(\mathbf{x})$ at discrete locations indicated as black or white circles. An easy way to detect and represent zero crossings is to look for adjacent pixel locations \mathbf{x}_i and \mathbf{x}_j where the sign changes value, i.e., $[S(\mathbf{x}_i) > 0] \neq [S(\mathbf{x}_j) > 0]$.

The sub-pixel location of this crossing can be obtained by computing the “ x -intercept” of the “line” connecting $S(\mathbf{x}_i)$ and $S(\mathbf{x}_j)$, as shown in Figure 4.34b,

$$\mathbf{x}_z = \frac{\mathbf{x}_i S(\mathbf{x}_j) - \mathbf{x}_j S(\mathbf{x}_i)}{S(\mathbf{x}_j) - S(\mathbf{x}_i)}. \quad (4.25)$$

Figure 4.34a shows the detected edgels as small oriented red line segments living on a grid *dual* to the original pixel grid. The orientation and strength of such edgels can be obtained by linearly interpolating the gradient values computed on the original pixel grid.

An alternative edgel representation can be obtained by linking adjacent edgels on the dual grid to form edgels that live *inside* each square formed by four adjacent pixels in the original pixel grid, as shown by the blue line segments in Figure 4.34a.⁴ The (potential) advantage of this representation is that the edgels now live on a grid offset by a $1/2$ -pixel from the original pixel grid, and are thus easier to store and access. [Note: This last sentence may not be that clear. Give more details in Exercise 4.8.] As before, the orientations and strengths of the edges can be computed by interpolating the gradient field or estimating these values from the difference of Gaussian image (see Exercise 4.8). [Note: Defer the discussion of the topology confusion to the linking section.]

In applications where the accuracy of the edge orientation is more important, higher-order steerable filters can be used (Freeman and Adelson 1991), §3.2.1. Such filters are more selective for more elongated edges, and also have the possibility of better modeling curve intersections because they can represent multiple orientations at the same pixel (Figure 3.15). Their disadvantage is that they are more computationally expensive to compute and the directional derivative of the edge strength does not have a simple closed form solution.⁵ [Note: I was going to say that you

⁴ This algorithm is a 2D version of the 3D *marching cubes* isosurface extraction algorithm [Note: Find reference to *marching cubes*].

⁵ In fact, the edge orientation can have a 180° ambiguity for “bar edges”, which makes the computation of zero crossings in the derivative more tricky.

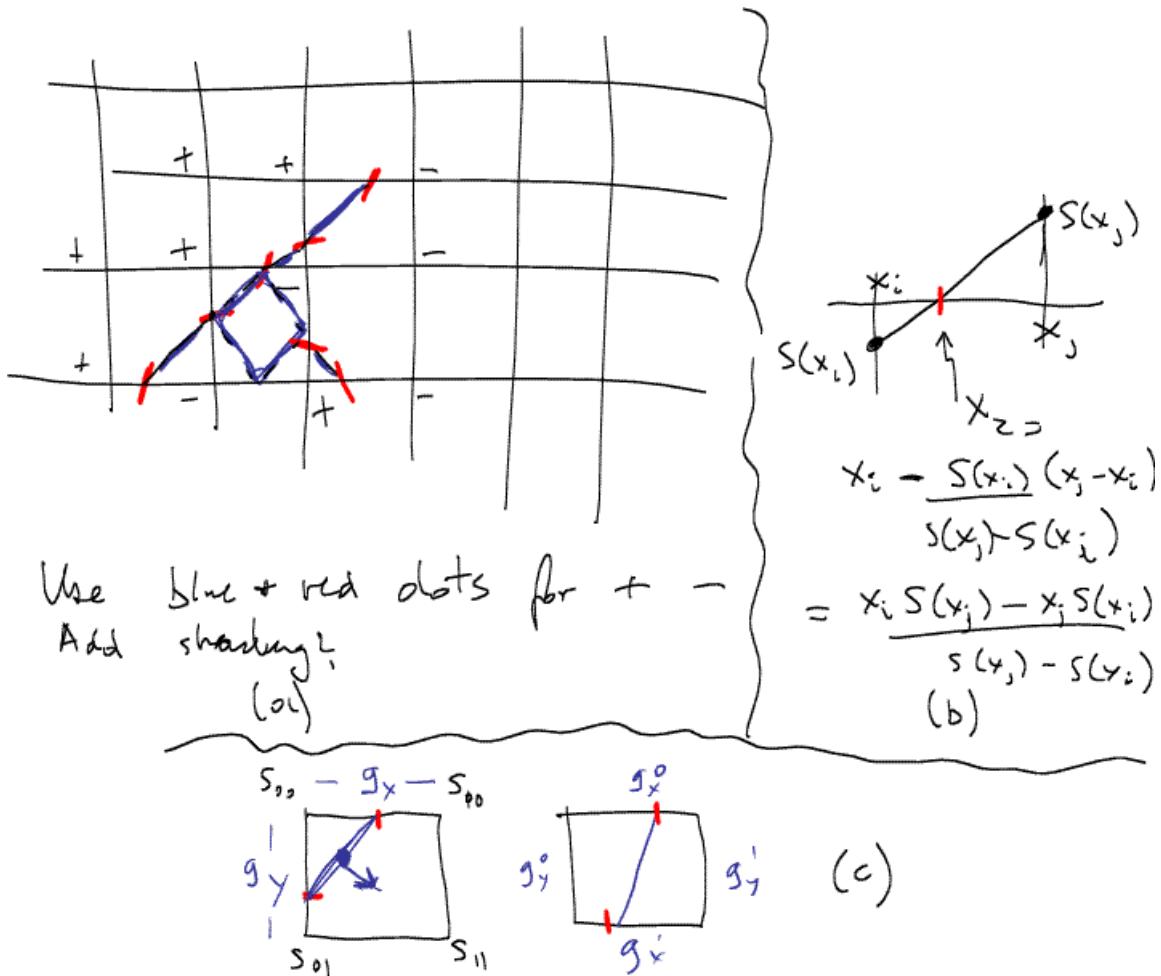


Figure 4.34: Detecting and linking zero crossings in a Laplacian of Gaussian image. The signed values along horizontal and vertical edges can be used to locate the edgels to sub-pixel precision. The zero crossing topology may be ambiguous when the positive and negative values alternate around a square.

[Note: Generate a proper figure]

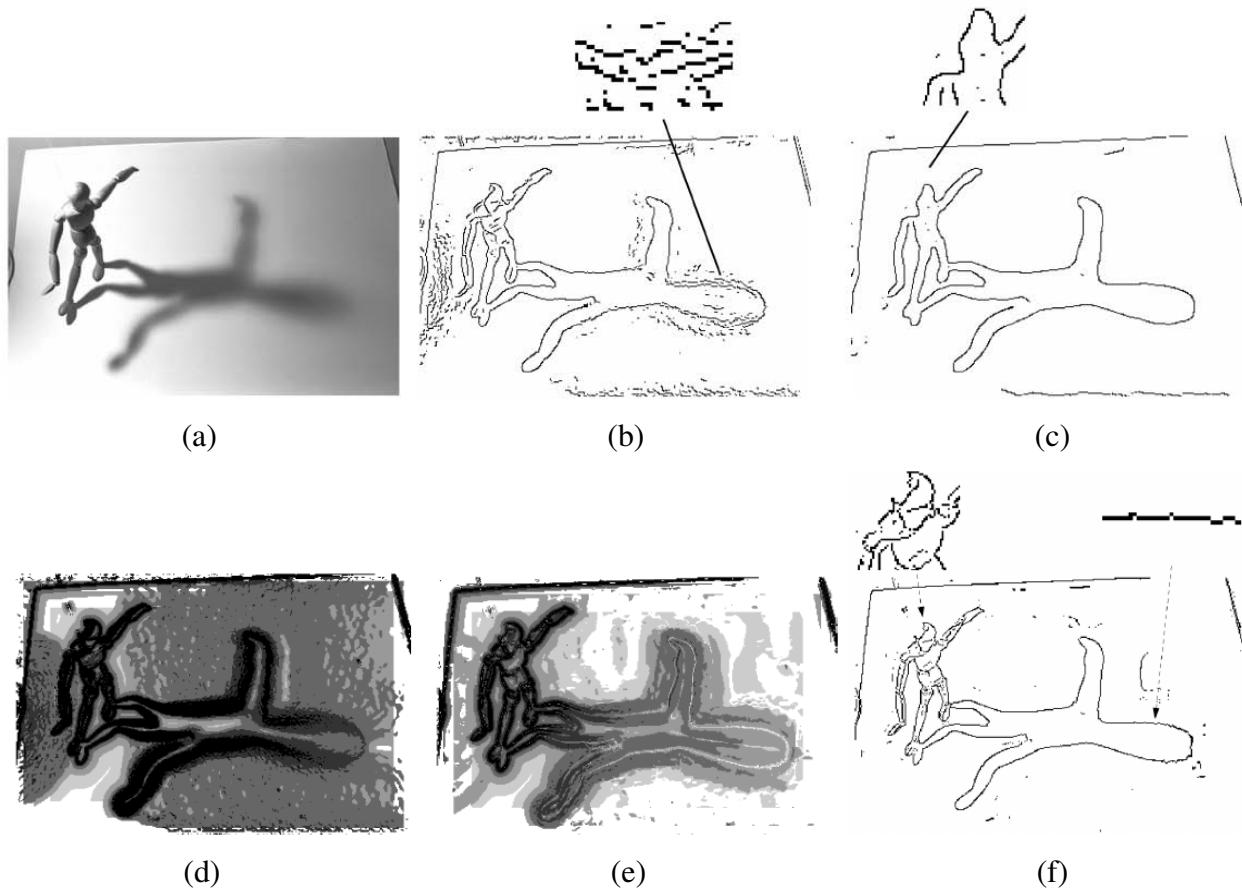


Figure 4.35: *Scale selection for edge detection (Elder and Zucker 1998)*. (a) original image; (b–c) Canny/Deriche edge detector tuned to the finer (mannequin) and coarser (shadow) scales; (d) minimum reliable scale for gradient estimation; (e) minimum reliable scale for second derivative estimation; (f) final detected edges.

cannot as easily find the maximum, but my CVisSteerableFilter code seems to do that. It does it by computing a strength and orientation field at each pixel, then taking directional derivatives (using central differences) of the strength field (see `find_maxima()`). During the zero crossing computation, inconsistent angles are checked for and the derivative values are negated, as necessary. Need to re-read (Freeman and Adelson 1991) carefully and revive the code so that I can generate some figures here...]

Scale selection and blur estimation

As I mentioned before, the derivative, Laplacian, and Difference of Gaussian filters (4.20–4.23) all require the selection of a spatial scale parameter σ . If we are only interested in detecting sharp

edges, the width of the filter can be determined from image noise characteristics (Canny 1986, Elder and Zucker 1998). However, if we want to detect edges that occur at different resolutions (Figures 4.35b–c), a *scale-space* approach that detects and then selects edges at different scales may be necessary (Witkin 1983, Lindeberg 1994, Lindeberg 1998a, Nielsen *et al.* 1997).

Elder and Zucker (1998) present a principled approach to solving this problem. Given a known image noise level, their technique computes for every pixel the minimum scale at which an edge can be reliably detected (Figure 4.35d). Their approach first computes gradients densely over an image by selecting among gradient estimates computed at different scales (based on their gradient magnitudes). It then performs a similar estimate of minimum scale for directed second derivatives, and then uses zero crossings of this latter quantity to robustly select edges. (Figures 4.35e–f). As an optional final step, the blur width of each edge can be computed from the distance between extrema in the second derivative response minus the width of the Gaussian filter.

Color edge detection

While most edge detection techniques have been developed for grayscale images, color images can provide additional information. For example, noticeable edges between *iso-luminant* colors (colors that have the same luminance) are useful cues, but will fail to be detected by grayscale edge operators.

One simple approach is to combine the outputs of grayscale detectors run on each color band separately.⁶ However, some care must be taken. For example, if we simply sum up the gradients in each of the color bands, the signed gradients may actually cancel each other! (Consider, for example a pure red-to-green edge.) We could also detect edges independently in each band and then take the union of these, but this might lead to thickened or doubled edges that are hard to link.

A better approach is to compute the *oriented energy* in each band (Morrone and Burr 1988, Perona and Malik 1990a), e.g., using a second order steerable filter §3.2.1 (Freeman and Adelson 1991), and to then sum up the orientation-weighted energies and find their joint best orientation. Unfortunately, the directional derivative of this energy may not have a closed form solution (as in the case of signed first order steerable filters), so a simple zero crossing-based strategy cannot be used. However, the technique described by Elder and Zucker (1998) can be used to numerically compute these zero crossings instead.

[Note: Get some additional citations, e.g., from Keith Price's bibliography at <http://iris.usc.edu/Vision-Notes/bibliography/edge267.html#TT1557> (see also misc/Keith Price

⁶ Instead of using the raw RGB space, a more perceptually uniform color space such as L*a*b* §2.3.2 can be used instead. When trying to match human performance (Martin *et al.* 2004), this makes sense. However, in terms of the physics of the underlying image formation and sensing, this may be questionable. [Note: Did Carol Nowak ever compare these?]

Bibliography Color Edge Detectors.mht) has some references. Also, did Nowak/Shafer do something like this?]

An alternative approach is to estimate local color statistics in half-discs around each pixel (Martin *et al.* 2004). This has the advantage that more sophisticated techniques can be used to compare regional statistics (e.g., 3D color histograms) and that additional measures, such as texture, can also be considered. Figure 4.36 shows the output of such detectors.

Combining edge feature cues

If the goal of edge detection is to match human *boundary detection* performance (Bowyer *et al.* 2001, Martin *et al.* 2004), as opposed to simply finding stable features for matching, even better detectors can be constructed by combining multiple low-level cues such as brightness, color, and texture.

Martin *et al.* (2004) describe a system that combines brightness, color, and texture edges to produce state-of-the-art performance on a database of hand-segmented natural color images (Martin *et al.* 2001). First, they construct and train separate oriented half-disc detectors for measuring significant differences in brightness (luminance), color (a^* and b^* channels, summed responses), and texture (un-normalized filter bank responses from (Malik *et al.* 2001)). (The training uses 200 labelled image, and testing is performed on a different set of 100 images.) Next, some of these responses are sharpened using a soft non-maximal suppression technique. Finally, the outputs of the three detectors are combined using a variety of machine learning techniques, among which, logistic regression is found to have the best speed-space-accuracy tradeoff. The resulting system (see Figure 4.36 for some visual examples) is shown to outperform previously developed techniques. In more recent work, Maire *et al.* (2008) improve on these results by combining the local appearance based detector with a *spectral* (segmentation-based) detector (Belongie and Malik 1998).

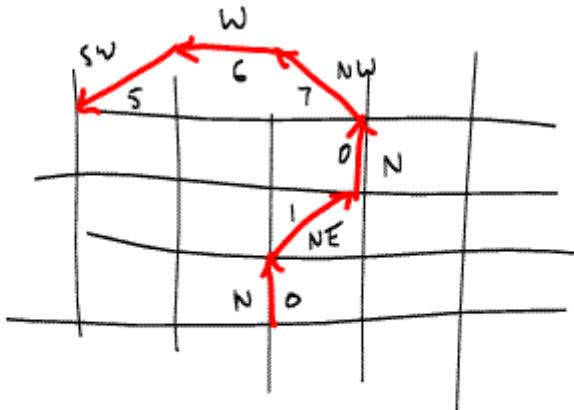
4.2.2 Edge linking

While isolated edges can be useful for a variety of applications such as line detection §4.3 and sparse stereo matching §10.2, they become even more useful when linked into continuous contours.

If the edges have been detected using zero crossings of some function, linking them up is straightforward, since adjacent edgels share common endpoints (Figure 4.34a). Linking the edgels into chains involves picking up an unlinked edgel and following its neighbors in both directions. Either a sorted list of edgels (sorted first say by x coordinates and then y coordinates) or a 2D array can be used to accelerate the neighbor finding. If edges were not detected using zero crossings, finding the continuation of an edgel can be tricky. In this case, comparing the orientation (and optionally phase) of adjacent edgels can be used for disambiguation. Ideas from connected com-



Figure 4.36: *Combined brightness/color/texture boundary detector* (Martin et al. 2004). Successive rows show the outputs of the brightness gradient (BG), color gradient (CG), texture gradient (TG), and combined (BG+CG+TG) detectors. The final row shows human-labelled boundaries derived from a database of hand-segmented images (Martin et al. 2001).



Chain code : 010765

Figure 4.37: *Chain code representation of a grid-aligned linked edge chain. The code is represented as a series of direction codes, e.g., 010765, which can further be compressed.*

ponent computation can also sometimes be used to make the edge linking process even faster (see Exercise 4.9).

Once the edgels have been linked into chains, we can apply an optional thresholding with hysteresis to remove low-strength contour segments (Canny 1986). [Note: Re-read Canny and describe this better.]

Linked edgel lists can be encoded more compactly using a variety of alternative representations. A *chain code* encodes a list of connected points lying on an \mathcal{N}_8 grid using a different 3-bit code corresponding to the eight cardinal directions (N, NE, E, SE, S, SW, W, NW) between a point and its successor (Figure 4.37). While this representation is more compact than the original edgel list (especially if predictive variable-length coding is used), it is not very suitable for further processing.

A more useful representation is the *arc-length parameterization* of a contour, $\mathbf{x}(s)$, where s denotes the arc length along a curve. Consider the linked set of edgels shown in Figure 4.38a. We start at one point (the solid dot at $(1.0, 0.5)$ in Figure 4.38a) and plot it at coordinate $s = 0$ (Figure 4.38b). The next point at $(2.0, 0.5)$ gets plotted at $s = 1$, and the next point at $(2.5, 1.0)$ gets plotted at $s = 1.7071$, i.e., we increment s by the length of each edge segment. The resulting plot can be resampled on a regular (say integral) s grid before further processing.

The advantage of the arc-length parameterization is that it makes matching and processing (e.g., smoothing) operations much easier. Consider the two curves describing similar shapes shown in Figure 4.39. To compare the curves, we first subtract the average values $\mathbf{x}_0 = \int_s \mathbf{x}(s)$ from each descriptor. Next, we rescale each descriptor so that s goes from 0 to 1 instead of 0 to S , i.e., we divide $\mathbf{x}(s)$ by S . Finally, we take the Fourier transform of each normalized descriptor,

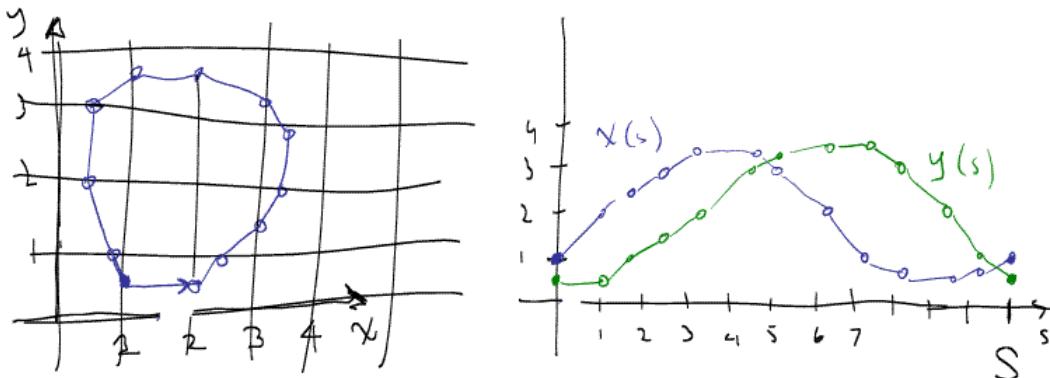


Figure 4.38: *Arc-length parameterization of a contour. Discrete points along the contour are first transcribed as (x, y) pairs along the arc length s . This curve can then be regularly re-sampled or converted into alternative (e.g., Fourier) representations.*

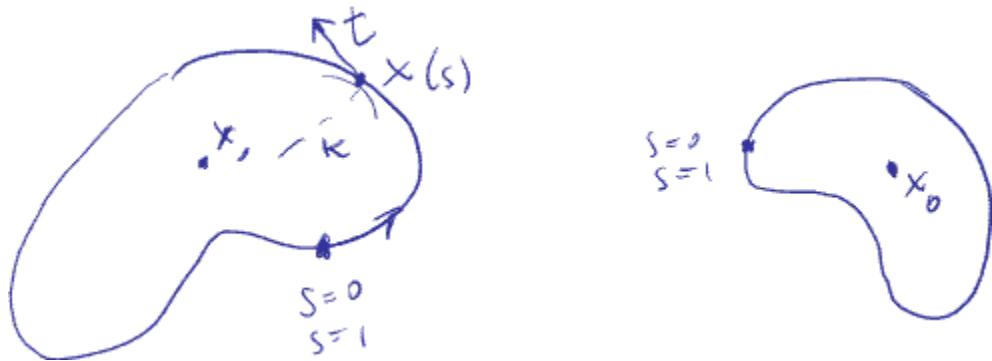


Figure 4.39: *Matching two contours using their arc-length parameterization. If both curves are normalized to unit length, $s \in [0, 1]$, they will have the same descriptor up to an overall “temporal” shift (due to different starting points for $s = 0$) and a phase (x - y) shift due to rotation.*

treating each $\mathbf{x} = (x, y)$ value as a complex number. If the original curves are the same (up to an unknown scale and rotation), the resulting Fourier transforms should differ only by a scale change in magnitude plus a constant phase shift, due to rotation, and a linear phase shift, due to different starting points for s (see Exercise 4.10). [Note: See if you can find a reference to this Fourier matching, either in a textbook or in some papers.]

[Note: Read the papers below before finishing off this section:]

Lowe (1989): as we filter a 2D curve with a Gaussian, estimate the second derivatives of the smoothed curves (x and y separately), and then add back a term to compensate for the shrinkage (which is a function of radius and hence second derivative). See Figure 4.40. **Taubin (1995)** extends this technique by replacing the Lowe’s offsetting step with one based on an additional (larger) smoothing...

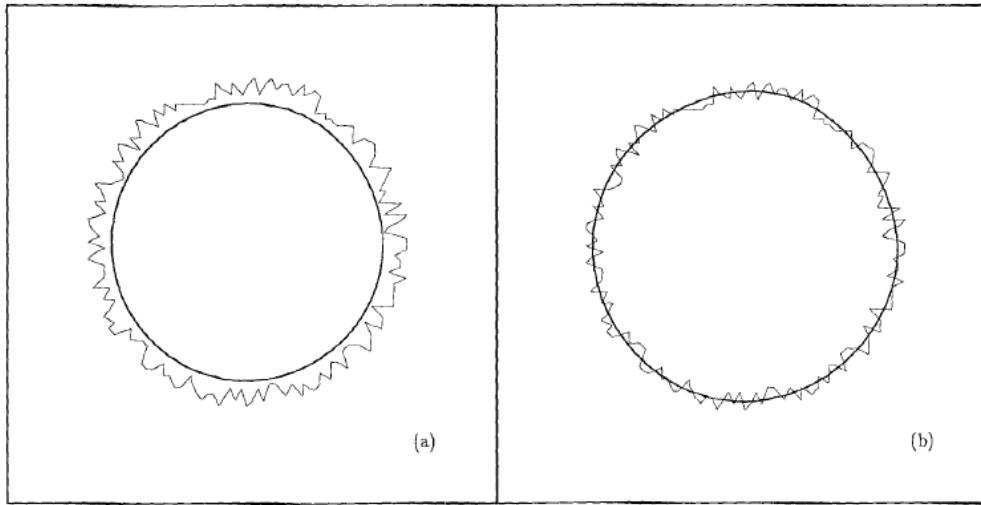


Figure 4.40: *Curve smoothing with a Gaussian kernel (Lowe 1988)*: (a) without shrinkage correction term; (b) with shrinkage correction term.

An alternative approach, based on selectively modifying different frequencies in a wavelet decomposition, is presented by Finkelstein and Salesin (1994). In addition to controlling shrinkage without affecting its “sweep”, wavelets allow the “character” of a curve to be interactively modified, as shown in Figure 4.41.

contour filtering: wavelets (Salesin and ?) and Taubin(?) or Lowe (?) - no shrinkage; Kimia’s descriptors and curve evolutions

Mention curvature-based (intrinsic) descriptors, independent of orientation (but dependent on scale)?

Also, shape contexts (Belongie *et al.* 2002).

Latest global contour grouping, completion, and junction detection (Maire *et al.* 2008): run normalized cuts (NC) on affinities derived from strongest Pb edge, then take gradients of eigenvectors to get “global Pb” and then later Pj (probability of junction)

Point to (Maire *et al.* 2008) for other recent edge detectors and contour linkers.

Meltzer and Soatto (2008) have a nice edge descriptor based on scale-space analysis followed by gradient histograms on each side of the edge. Can be used for wide baseline stereo.

4.2.3 Application: Edge editing and enhancement

While edges can serve as components for object recognition or features for matching, they can also be used directly for image editing.

In fact, if the edge magnitude and blur estimate are kept along with each edge, a visually similar

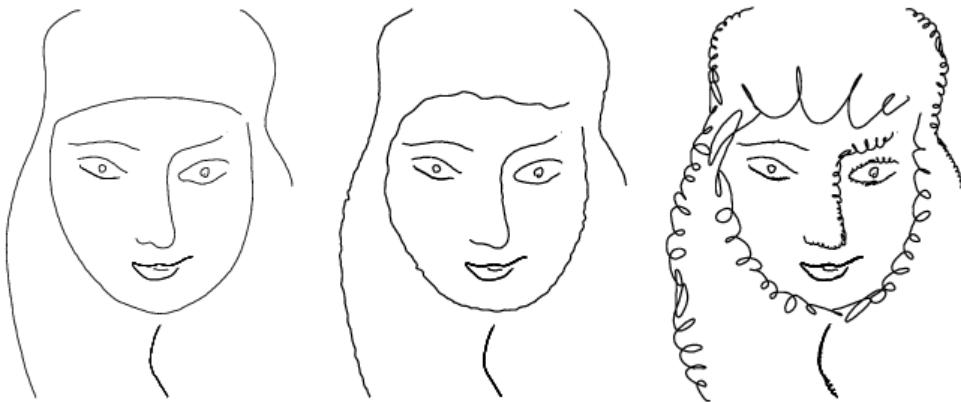


Figure 4.41: *Changing the character of a curve without affecting its sweep (Finkelstein and Salesin 1994): higher frequency wavelets can be replaced with exemplars from a style library to effect different local appearances.*

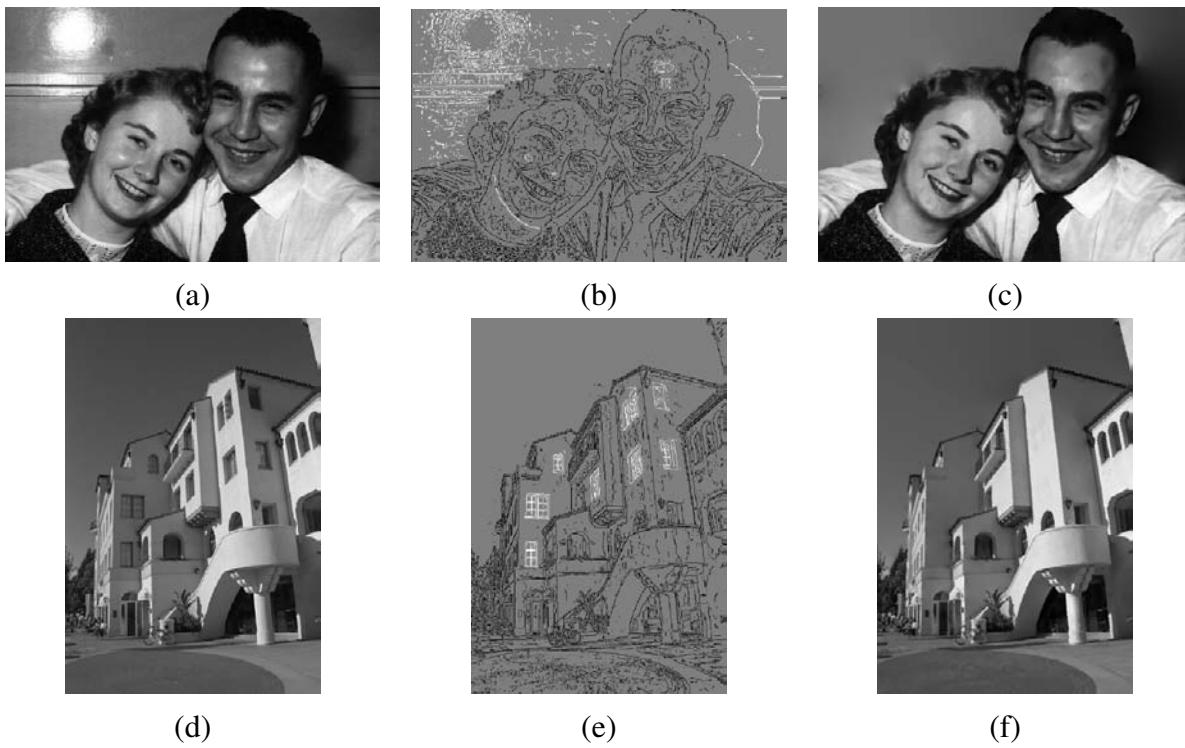


Figure 4.42: *Image editing in the contour domain (Elder and Goldberg 2001).* (a) & (d): original images; (b) & (e): extracted edges (edges to be deleted are marked in white); (c) & (f): reconstructed edited images.

image can be reconstructed from this information (Elder 1999). Based on this principle, Elder and Goldberg (2001) propose a system for “Image Editing in the Contour Domain”. Their system allows users to selectively remove edges corresponding to unwanted features such as specularities, shadows, or distracting visual elements. After reconstructing the image from the remaining edges, the undesirable visual features have been removed (Figure 4.42).

Another potential application is to enhance perceptually salient edges while simplifying the underlying image to produce a cartoon-like or “pen-and-ink” like stylised image (DeCarlo and Santella 2002). This application is discussed in more detail in the section on non-photorealistic rendering §9.6.

4.3 Lines

While edges and general curves are suitable for describing the contours of natural objects, the man-made worlds is full of straight lines. Detecting and matching these lines can be useful in a variety of applications, including architectural modeling, pose estimation in urban environments, and the analysis of printed document layouts.

[Note: Add a figure that shows each of the 3 cases (successive approx., Hough, VP) below?]

In this section, I present some techniques for extracting *piecewise linear* descriptions from the curves computed in the previous section. I begin with some algorithms for approximating a curve as a piecewise-linear polyline. I then describe the *Hough transform*, which can be used to group edgels into line segments even across gaps and occlusions. Finally, I describe how 3D lines with common *vanishing points* can be grouped together. These vanishing points can be used to calibrate a camera and to determine its orientation relative to a rectihedral scene, as described in §5.3.2.

4.3.1 Successive approximation

As we saw in the previous section 4.2.2, describing a curve as a series of 2D locations $\mathbf{x}_i = \mathbf{x}(s_i)$ provides a general representation suitable for matching and further processing. In many applications, however, it is preferable to approximate such a curve with a simpler representation, e.g., as a piecewise-linear polyline or as a B-spline curve (Farin 1996), as shown in Figure 4.43.

Many techniques have been developed over the years to perform this approximation.

[Note: Re-read some of the literature, including (Szeliski and Ito 1986) to get this right, then finish this off:]

Simplify linked edge structures (contours) into polylines; could also go in the curves section...

An alternative approach is bottom-up merge: merge adjacent edges into line or quadratic approximations, keep merging, or link if cannot merge because model does not fit.

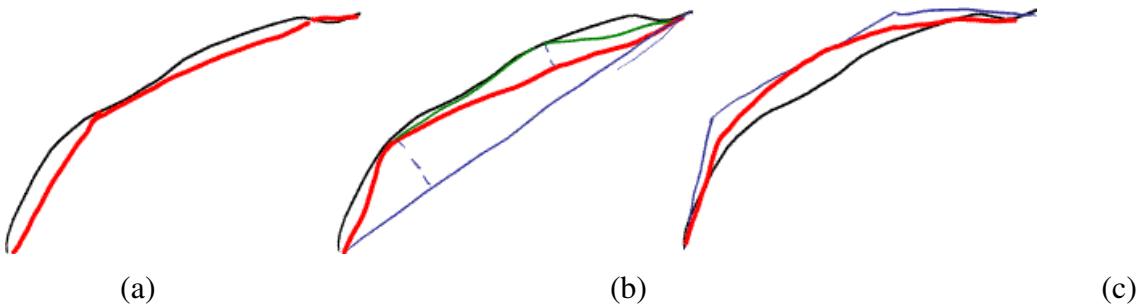


Figure 4.43: Approximating a curve (shown in black) as a polyline or B-spline: (a) original curve and a polyline approximation shown in red; (b) successive approximation by maximum chord [Note: right?]; (c) B-spline approximation, with the control net shown in blue and the B-spline shown in red.

[Note: Visio does this when you draw a free-form curve: if you hover over the (smoothed) curve, you can see the Bezier control points.]

Use a multi-res quad structure to quickly index neighbors for possible merging (level depends on current length of curve).

Mention B-spline representations (Figure 4.43c).

4.3.2 Hough transforms

While curve approximation with polylines can often lead to successful line extraction, lines in the real world are sometimes broken up into disconnected components or made up of many collinear line segments (Figure 4.44). In many cases, it is desirable to group such collinear line segments into extended lines. As a further processing stage, described in the next section §4.3.3, we can then group such lines into collections with common vanishing points.

The Hough transform, named after its original inventor (Hough 1962), is a well-known technique for having edges “vote” for plausible line locations (Duda and Hart 1972, Ballard 1981, Illingworth and Kittler 1988) [Note: Add more references: (Forsyth and Ponce 2003, Fitting Chapter); (Jähne 1997);] In its original formulation (Figure 4.45), each edge point votes for *all* possible lines passing through it, and lines corresponding to high *accumulator* or *bin* values are examined for potential line fits.⁷ Unless the points on a line are truly punctate, a better approach (in my experience) is to use the local orientation information at each edgel to vote for a *single* accumulator cell (Figure 4.46), as described below. A hybrid strategy, where each edgel votes for a number of possible orientation/location pairs centered around the estimate orientation,

⁷ The Hough transform can also be *generalized* to look for other geometric features such as circles (Ballard 1981), but I do not cover such extensions in this book.

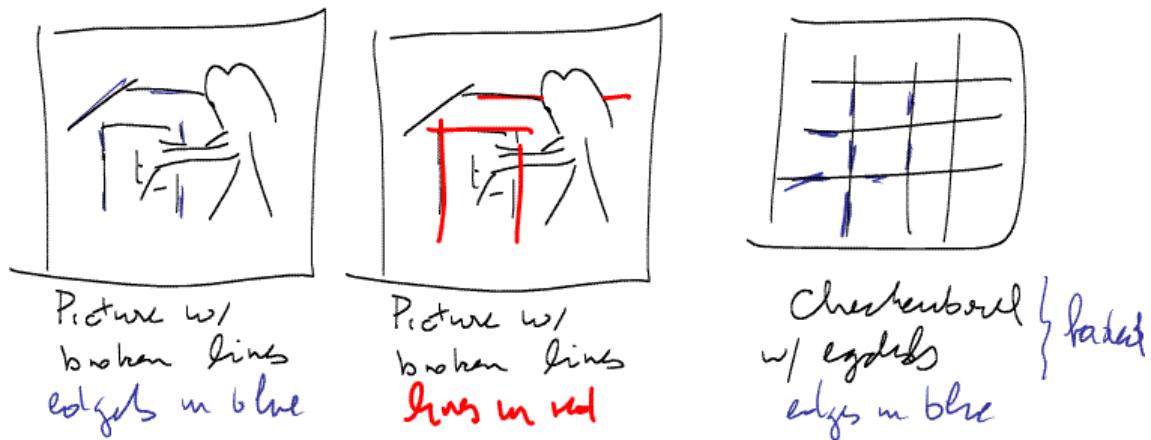


Figure 4.44: Line fitting to discontinuous edge segments: (a) image with clutter; (b) fitted lines; (c) checkerboard image with missing edgels at corners.

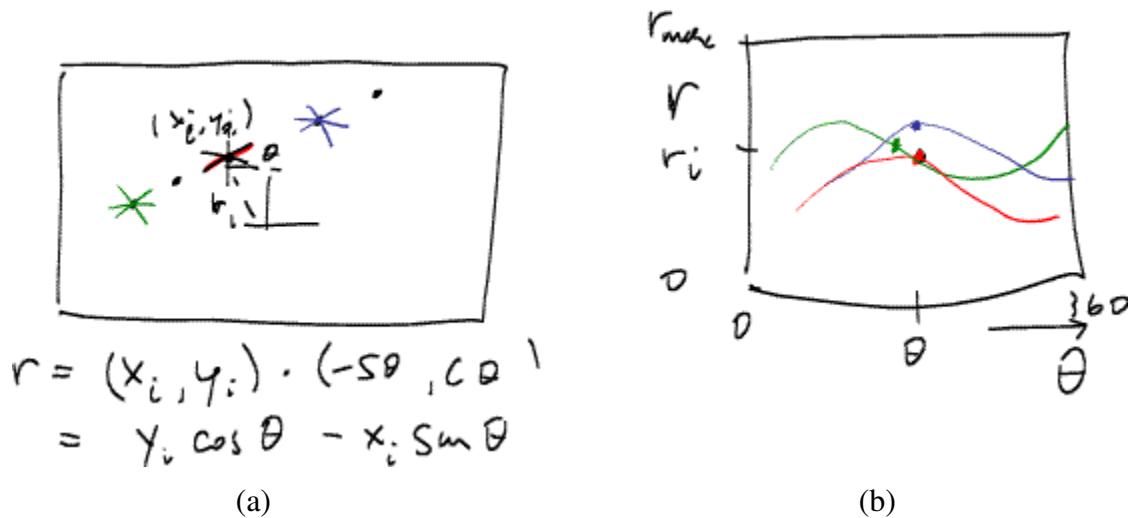


Figure 4.45: Original Hough transform: (a) each point votes for a complete family of potential lines $r_i(\theta) = x_i \cos \theta + y_i \sin \theta$; (b) each pencil of lines sweeps out a sinusoid in (r, θ) ; their intersection provides the desired line equation.

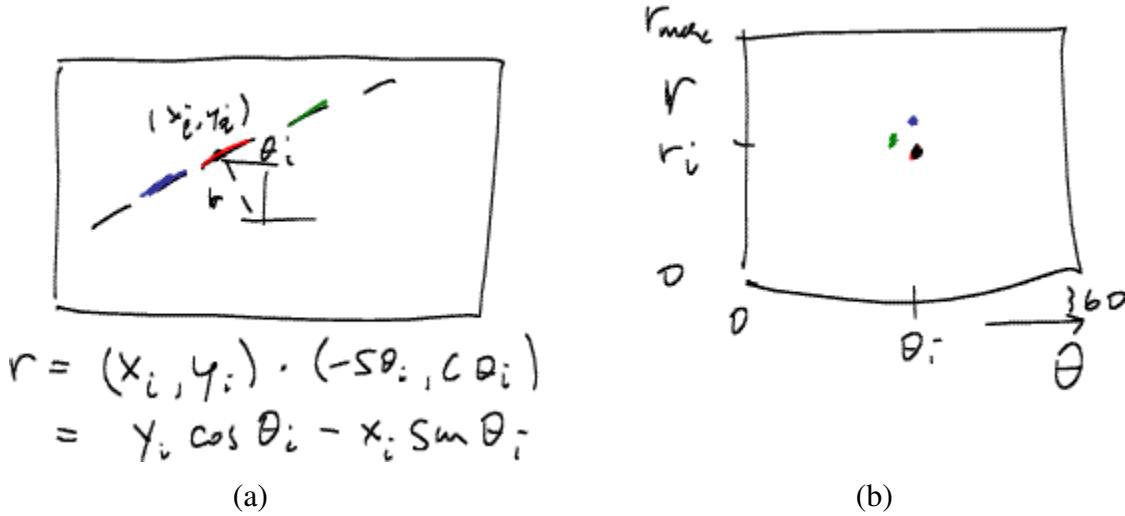


Figure 4.46: *Hough transform*: (a) an edgel re-parameterized in polar (r, θ) coordinates, with $\hat{\mathbf{n}}_i = (\cos \theta_i, \sin \theta_i)$ and $r_i = \hat{\mathbf{n}}_i \cdot \mathbf{x}_i$; (b) (r, θ) accumulator array, showing the votes for the three edgels marked in red, green, and blue.

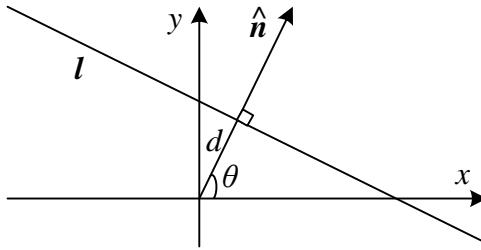


Figure 4.47: *2D line equation expressed in terms of the normal $\hat{\mathbf{n}}$ and distance to the origin d* (copied from Figure 2.2).

may be desirable in some cases.

Before we can vote for line hypotheses, we must first choose a suitable representation. Figure 4.47 (copied from Figure 2.2) shows the normal-distance $(\hat{\mathbf{n}}, d)$ parameterization for a line. Since lines are made up of edge segments, we adopt the convention that the line normal $\hat{\mathbf{n}}$ points in the same direction (has the same signs) as the image gradient $\mathbf{J}(\mathbf{x}) = \nabla I(\mathbf{x})$ (4.19). To obtain a minimal two-parameter representation for lines, we convert the normal vector into an angle

$$\theta = \tan^{-1} n_y / n_x, \quad (4.26)$$

as shown in Figure 4.47. The range of possible (θ, d) values is $[-180^\circ, 180^\circ] \times [-\sqrt{2}, \sqrt{2}]$, assuming that we are using normalized pixel coordinates (2.60) that lie in $[-1, 1]$. The number of bins to use along each axis depends on the accuracy of the position and orientation estimate available

at each edgel and the expected line density, and is best set experimentally with some test runs on sample imagery.

Given the line parameterization, the Hough transform proceeds as follows:

1. Clear the accumulator array.
2. For each detected edgel at location (x, y) and orientation $\theta = \tan^{-1} n_y/n_x$, compute the value of

$$d = x n_x + y n_y \quad (4.27)$$

and increment the accumulator corresponding to (θ, d) .

3. Find the peaks in the accumulator corresponding to lines.
4. Optionally re-fit the lines to the constituent edgelets.

Note that the original formulation of the Hough transform, which assumed no knowledge of the edgel orientation θ , has an additional loop inside of step 2 that iterates over all possible values of θ and increments a whole series of accumulators.

There are a lot of details in getting the Hough transform to work well, but these are best worked out by writing an implementation and testing it out on sample data. Exercise 4.14 describes some of these steps in more detail, including using edge segment lengths and/or strength during the voting process, keeping a list of constituent edgelets in the accumulator array for easier post-processing, and optionally combining edges of different “polarity” into the same line segments.

An alternative to the 2-D polar representation (θ, d) representation for lines is to use the full 3-D $\mathbf{m} = (\hat{\mathbf{n}}, d)$ line equation, projected onto the unit sphere. While the sphere can be parameterized using spherical coordinates (2.8),

$$\hat{\mathbf{m}} = (\cos \theta \cos \phi, \sin \theta \cos \phi, \sin \phi), \quad (4.28)$$

this does not uniformly sample the sphere and still requires the use of trigonometry.

An alternative representation can be obtained by using a *cube map*, i.e., projecting \mathbf{m} onto the face of a unit cube (Figure 4.48a). To compute the cube map coordinate of a 3-D vector \mathbf{m} , first find the largest (absolute value) component of \mathbf{m} , i.e., $m = \pm \max(|n_x|, |n_y|, |d|)$, and use this to select one of the six cube faces. Next, divide the remaining two coordinates by m and use these as indices into the cube face. While this avoids the use of trigonometry, it does require some decision logic.

One advantage of using the cube map, first pointed out by Tuytelaars *et al.* (1997) in their paper on the *Cascaded Hough Transform*, is that all of the lines passing through a point correspond to line segments on the cube faces, which is useful if the original (full voting) variant of the Hough

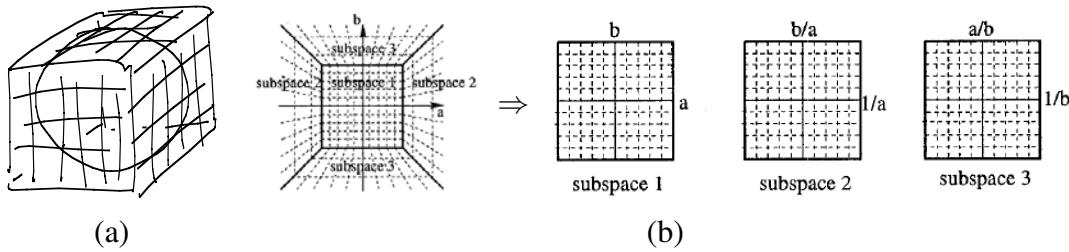


Figure 4.48: *Cube map representation for line equations and vanishing points: (a) a cube map surrounding the unit sphere; (b) projecting the half-cube onto three sub-spaces (Tuytelaars et al. 1997).*

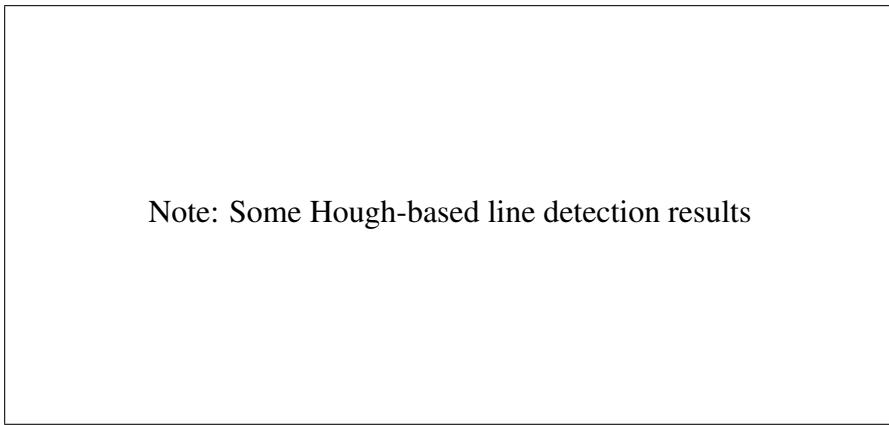


Figure 4.49: Some Hough-based line detection results

[Note: Show both accumulator array and overlaid lines]

transform is being used. In their work, they represent the line equation as $ax + b + y = 0$, which does not treat the x and y axes symmetrically. Note that if we restrict $d \geq 0$ by ignoring the polarity of the edge orientation (gradient sign), we can use a half-cube instead, which can be represented using only three cube faces (Figure 4.48b) (Tuytelaars *et al.* 1997).

Figure 4.49 shows the result of applying a Hough transform line detector to an image. As you can see, the technique successfully finds a lot of the extended lines in the image, but also occasionally find spurious lines due to the coincidental alignment of unrelated edgels. One way to reduce the number of spurious lines is to look for common vanishing points, as described in the next section §4.3.3.

RANSAC-base line detection. Another alternative to the Hough transform is the RAnDom SAMple Consensus (RANSAC) algorithm described in more detail in §5.1.5. In brief, RANSAC randomly chooses pairs of edgels to form a line hypothesis and then tests how many other edgels fall onto this line. (If the edge orientations are accurate enough, a single edgel can produce this

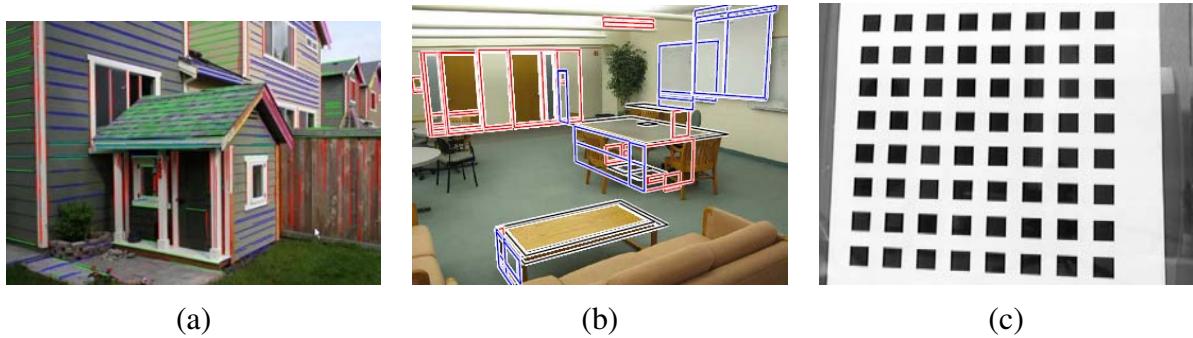


Figure 4.50: *Examples of real-world vanishing points: (a) architecture (Sinha et al. 2008), (b) furniture (Mičušík et al. 2008), and (c) calibration patterns (Zhang 1999).*

hypothesis.) Lines with sufficiently large numbers of *inliers* (matching edgels) are then selected as the desired line segments.

An advantage of RANSAC is that no accumulator array is needed and so the algorithm can be more space efficient (and potentially less prone to the choice of bin size). The disadvantage is that many more hypotheses may be generated and tested than those obtained by finding peaks in the accumulator array.

In general, there is no clear consensus on which line estimation technique performs the best. It is therefore a good idea to think carefully about the problem at hand and to implement several approaches (successive approximation, Hough, and/or RANSAC) to determine the one that works best for your application.

4.3.3 Vanishing points

In many scenes, structurally important lines have the same vanishing point because they have they are parallel. Examples of such lines are horizontal and vertical building edges, zebra crossings, railway tracks, the edges of furniture such as tables and dressers, and of course, the ubiquitous calibration pattern (Figure 4.50). Finding the vanishing points common to such line sets can help refine their position in the image, and, in certain cases, help determine the intrinsic and extrinsic orientation of the camera §5.3.2.

Over the years, a large number of techniques have been developed for finding vanishing points, including (Quan and Mohr 1989, Collins and Weiss 1990, Brillaut-O’Mahoney 1991, McLean and Kotturi 1995, Becker and Bove 1995, Shufelt 1999, Tuytelaars *et al.* 1997, Schaffalitzky and Zisserman 2000, Antone and Teller 2000, Rother 2002, Košecká and Zhang 2005)—see some of the more recent papers for additional references. In this section, I present a simple Hough technique based on having line pairs vote for potential vanishing point locations, followed by a robust least squares fitting stage. For alternative approaches, please see some of the more recent papers listed

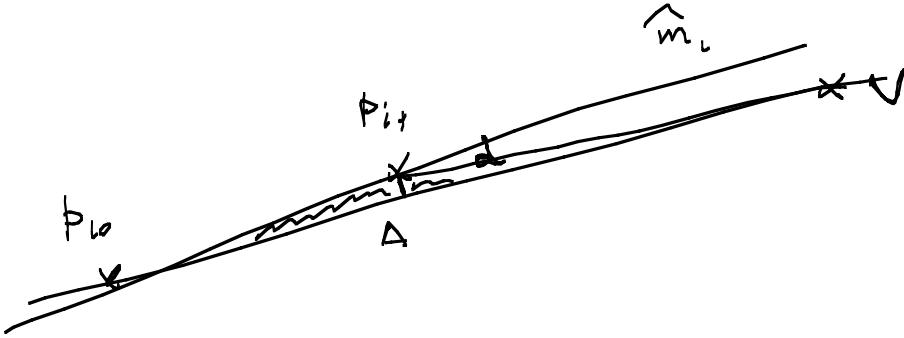


Figure 4.51: *Triple product of the line segments endpoints p_{i0} and p_{i1} and the vanishing point v . The area A is proportional to the perpendicular distance d_1 and the distance between the other endpoint p_{i0} and the vanishing point.*

above.

The first stage in my vanishing point detection algorithm uses a Hough transform to accumulate votes for likely vanishing point candidates. As with line fitting, one possible approach is to have each line vote for *all* possible vanishing point directions, either using a cube map (Tuytelaars *et al.* 1997, Antone and Teller 2000) or a Gaussian sphere (Collins and Weiss 1990), optionally using knowledge about the uncertainty in the vanishing point location to perform a weighted vote (Collins and Weiss 1990, Brillaut-O'Mahoney 1991, Shufelt 1999). My preferred approach is to use pairs of detected line segments to form candidate VP locations. Let \hat{m}_i and \hat{m}_j be the (unit norm) line equations for a pair of line segments and l_i and l_j be their corresponding segment lengths. The location of the corresponding vanishing point hypothesis can be computed as

$$\mathbf{v}_{ij} = \hat{m}_i \times \hat{m}_j \quad (4.29)$$

and the corresponding weight set to

$$w_{ij} = \|\mathbf{v}_{ij}\| l_i l_j. \quad (4.30)$$

This has the desirable effect of downweighting (near-)collinear line segments and short line segments. The Hough space itself can either be represented using spherical coordinates (4.28) or as a cube map (Figure 4.48a).

Once the Hough accumulator space has been populated, peaks can be detected in a manner similar to that previously discussed for line detection. Given a set of candidate line segments that voted for a vanishing point, which can optionally be kept as a list at each Hough accumulator cell, I then use a robust least squares fit to estimate a more accurate location for each vanishing point.

Consider the relationship between the two line segment endpoints $\{p_{i0}, p_{i1}\}$ and the vanishing point v , as shown in Figure 4.51. The area A of the triangle given by these three points, which is

the magnitude of their triple product

$$A_i = |(\mathbf{p}_{i0} \times \mathbf{p}_{i1}) \cdot \mathbf{v}|, \quad (4.31)$$

is proportional to the perpendicular distance d_1 between each endpoint and the line through \mathbf{v} and the other endpoint (as well as the distance between \mathbf{p}_{i0} and \mathbf{v}). Assuming that the accuracy of a fitted line segment is proportional to the endpoint accuracy (Ex 4.15), this therefore serves as a reasonable metric for how well a vanishing point fits a set of extracted lines. A robustified least squares estimate (§B.3) for the vanishing point can therefore be written as

$$\mathcal{E} = \sum_j \rho(\mathcal{A}_j) = \mathbf{v}^\top \left(\sum_j \exists_j(\mathcal{A}_j) \mathbf{m}_j \mathbf{m}_j^\top \right) \mathbf{v} = \mathbf{v}^\top \mathbf{M} \mathbf{v}, \quad (4.32)$$

where $\mathbf{m}_i = \mathbf{p}_{i0} \times \mathbf{p}_{i1}$ is the segment line equation weighted by its length l_i , and $w_i = \rho'(A_i)/A_i$ is the *influence* of each robustified (re-weighted) measurement on the final error (§B.3). Notice how this metric is closely related to the original formula for the pairwise weighted Hough transform accumulation step. The final desired value for \mathbf{v} is computed as the least eigenvector of \mathbf{M} .

While the technique described above proceeds in two discrete stages, better results may be obtained by alternating between assigning lines to vanishing points and refitting the vanishing point locations (Antone and Teller 2000, Košecká and Zhang 2005). The results of detecting individual vanishing points can also be made more robusts by simultaneously searching for pairs or triplets of mutually orthogonal vanishing points (Shufelt 1999, Antone and Teller 2000, Rother 2002, Sinha *et al.* 2008). Some results of such vanishing point detection algorithms can be seen in Figure 4.50.

4.3.4 Application: Rectangle detection

Once sets of mutually orthogonal vanishing points have been detected, it now becomes possible to search for 3D rectangular structures in the image (Figure 4.52). Over the last decade, a variety of techniques have been developed to find such rectangles, primarily focused on architectural scenes (Košecká and Zhang 2005, Han and Zhu 2005, Shaw and Barnes 2006, Mičušík *et al.* 2008, Schindler *et al.* 2008).

After detecting orthogonal vanishing directions, Košecká and Zhang (2005) refine the fitted line equations, search for corners near line intersections, and then verify rectangle hypotheses by rectifying the corresponding patches and looking for a preponderance of horizontal and vertical edges (Figures 4.52a–b). In follow-on work, Mičušík *et al.* (2008) use a Markov Random Field (MRF) to disambiguate between potentially overlapping rectangle hypotheses. They also use a plane sweep algorithm to match rectangles between different views (Figures 4.52d–f).

A different approach is proposed by Han and Zhu (2005), who use a grammar of potential rectangle shapes and nesting structures (between rectangles and vanishing points) to infer the most

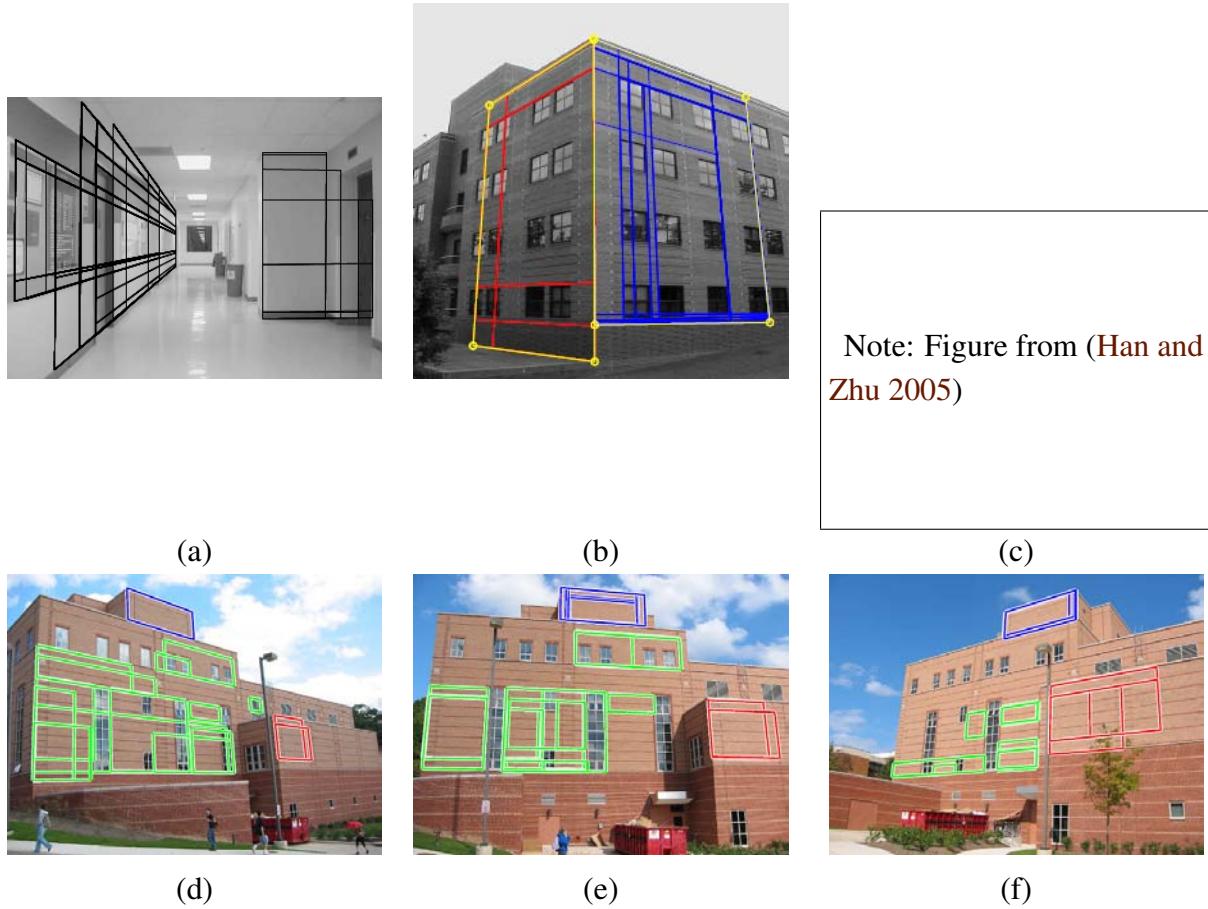


Figure 4.52: Examples of rectangle detection: (a) indoor corridor and (b) building exterior with grouped facades (Košecká and Zhang 2005); (c) grammar-based recognition (Han and Zhu 2005); (d–e) rectangle matching using plane sweep (Mičušík et al. 2008).

likely assignment of line segments to rectangles (Figure 4.52c). [Note: Expand this a little after re-reading the paper and fixing up the figure.]

4.4 Active contours

While lines, vanishing points, and rectangles are commonplace in the man-made world, curves corresponding to object boundaries are even more common, especially in the natural environment. In this section, I describe three related approaches to locating such boundary curves in images.

The first, originally called *snakes* by its inventors, (Kass *et al.* 1988) §4.4.1, is an energy-minimizing two-dimensional spline curve that evolves (moves) towards image features such strong edges. The second, *intelligent scissors* (Mortensen and Barrett 1995) §4.4.2, allow the user to

sketch in real time a curve that clings to object boundaries. Finally, level set techniques §4.4.3 evolve the curve as the zero-set of a *characteristic function*, which allows them to easily change topology and incorporate region-based statistics.

All three of these are examples of *active contours* (Blake and Isard 1998, Mortensen 1999), since these boundary detectors iteratively move towards their final solution under the (potential) combination of image and user-guided forces.

4.4.1 Snakes

Snakes are a two-dimensional (range) generalization of the 1-D energy minimizing splines first introduced in §3.6.1 (3.91–3.92),

$$\mathcal{E}_{\text{int}} = \int \alpha(s) \|\mathbf{f}_s(s)\|^2 + \beta(s) \|\mathbf{f}_{ss}(s)\|^2 ds, \quad (4.33)$$

where s is the arc-length along the curve $\mathbf{f}(s) = (x(s), y(s))$ and $\alpha(s)$ and $\beta(s)$ are first and second-order continuity weighting functions analogous to the $s(x, y)$ and $c(x, y)$ terms introduced in (3.99–3.100). We can discretize this energy by sampling the initial curve position evenly along its length (Figure 4.38) to obtain

$$\begin{aligned} E_{\text{int}} &= \sum_i \alpha(i) \|f(i+1) - f(i)\|^2 / h^2 \\ &\quad + \beta(i) \|f(i+1) - 2f(i) + f(i-1)\|^2 / h^4, \end{aligned} \quad (4.34)$$

where h is the step size, which can be neglected if we resample the curve along its arc-length after each iteration.

In addition to this *internal* spline energy, a snake simultaneously minimizes external image-based and constraint-based potentials. The image-based potentials are the sum of several terms

$$\mathcal{E}_{\text{image}} = w_{\text{line}} \mathcal{E}_{\text{line}} + w_{\text{edge}} \mathcal{E}_{\text{edge}} + w_{\text{term}} \mathcal{E}_{\text{term}}, \quad (4.35)$$

where the *line* term attracts the snake to dark ridges, the *edge* term attracts it to strong gradients (edges), and the *term* term attracts it to line terminations. In practice, most subsequent systems only use the edge term, which can either be directly proportional to the image gradients,

$$E_{\text{edge}} = \sum_i -\|\nabla I(\mathbf{f}(i))\|^2, \quad (4.36)$$

or to a smoothed version of the image Laplacian,

$$E_{\text{edge}} = \sum_i -|(G_\sigma * \nabla^2 I)(\mathbf{f}(i))|^2. \quad (4.37)$$

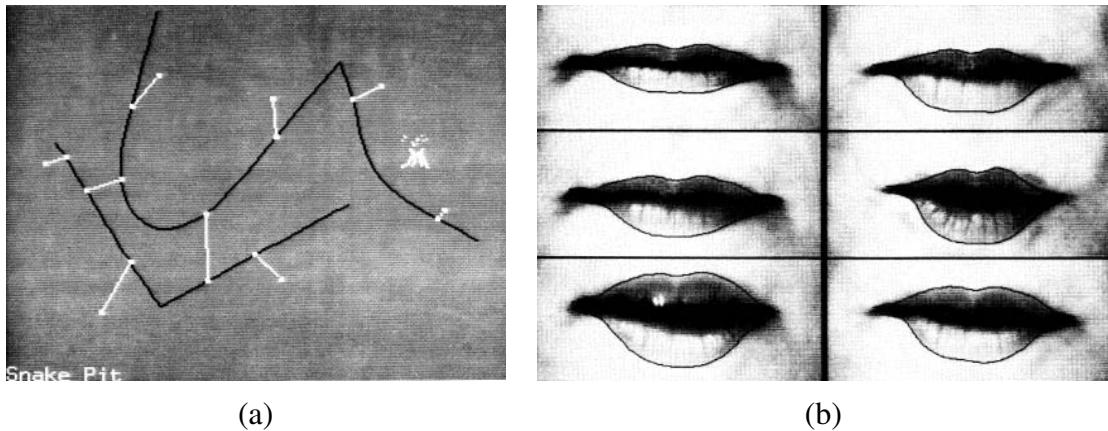


Figure 4.53: *Snakes* (Kass et al. 1988): (a) the “snake pit” for interactively controlling shape; (b) lip tracking.

People also sometimes extract edges and then use a distance map to the edges as an alternative to these two originally proposed potentials.

In interactive applications, a variety of user-placed constraints can also be added, e.g., attractive (spring) forces towards anchor points,

$$E_{\text{spring}} = k_i \|\mathbf{f}(i) - \mathbf{d}(i)\|^2, \quad (4.38)$$

as well as repulsive $1/r$ (“volcano”) forces (Figure 4.53a). As the snakes evolve by minimizing their energy, they often “wiggle” and “slither”, which accounts for their popular name. Figure 4.53b shows snakes being used to track a person’s lips.

Because regular snakes have a tendency to shrink (Exercise 4.18), it is usually better to initialize them by drawing the snake outside the object of interest to be tracked. Alternatively, an expansion *ballooning* force can be added to the dynamics (Cohen and Cohen 1993), essentially moving each point outwards along its normal.

To efficiently solve the sparse linear system arising from snake energy minimization, a sparse direct solver §A.4 can be used, since the linear system is essentially penta-diagonal.⁸ Snake evolution is usually implemented as an alternation between this linear system solution and the linearization of non-linear constraints such as edge energy. A more direct way to find a global energy minimum is to use dynamic programming (Amini et al. 1990), but this is not often used in practice, since it’s been superseeded by even more efficient and/or interactive algorithms such as intelligent scissors §4.4.2 and Grab Cut §4.5.4.

⁸ A closed snake has a Toeplitz matrix form, which can still be factored and solved in $O(n)$ time.

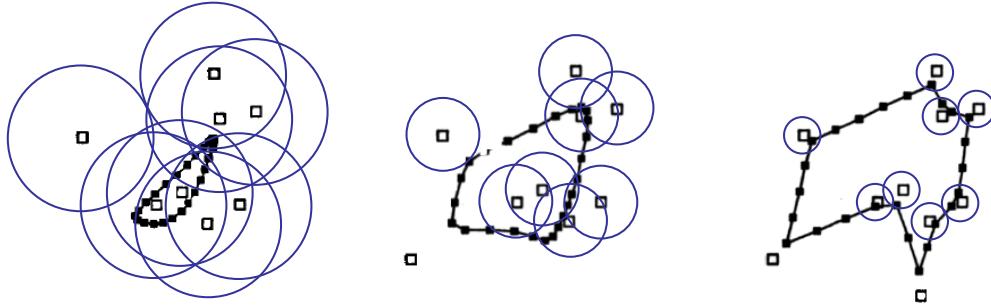


Figure 4.54: *Elastic net*: The open squares indicate the cities and the closed squares linked by straight line segments are the tour points. The blue circles indicate the approximate extent of the attraction force of each city, which is reduced over time. Under the Bayesian interpretation of the elastic net, the blue circles correspond to one standard deviation of the circular Gaussian that generate each city from some unknown tour point.

[Note: Redraw this better by hand]

Elastic nets and slippery springs

An interesting variant on snakes, first proposed by Durbin and Willshaw (1987) and later reformulated in an energy minimizing framework by Durbin *et al.* (1989) is the *elastic net* formulation of the Traveling Salesman Problem (TSP). Recall that in a TSP, the salesman must visit each city once while minimizing the total distance traversed. A snake that is constrained to pass through each city could solve this problem (without any optimality guarantees), but it is impossible to tell ahead of time which snake control point should be associated with each city.

Instead of having a fixed constraint between snake nodes and cities as in (4.38), a city is assumed to pass near *some* point along the tour (Figure 4.54). In a probabilistic interpretation, each city is generated as a *mixture* of Gaussians centered at each tour point,

$$p(\mathbf{d}(j)) = \sum_i p_{ij} \text{ with } p_{ij} = e^{-d_{ij}^2/(2\sigma^2)} \quad (4.39)$$

where σ is the standard deviation of the Gaussian and

$$d_{ij} = \|\mathbf{f}(i) - \mathbf{d}(j)\|, \quad (4.40)$$

is the Euclidean distance between a tour point i and a city j . The corresponding data fitting energy (negative log likelihood) is

$$E_{\text{slippery}} = - \sum_j \log p(\mathbf{d}(j)) = - \sum_j \log \left[\sum_i e^{-\|\mathbf{f}(i) - \mathbf{d}(j)\|^2/2\sigma^2} \right]. \quad (4.41)$$

This energy derives its name from the fact that unlike a regular spring, which couples a given snake point to a given constraint (4.38), this alternative energy defines a *slippery spring* that allows the

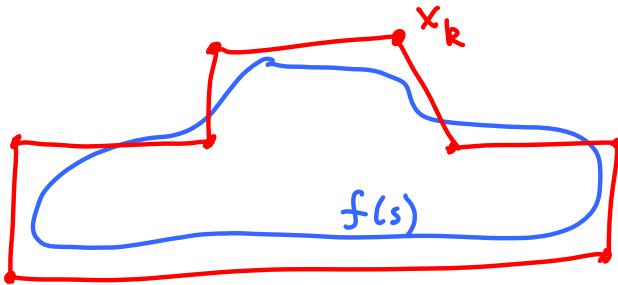


Figure 4.55: *B-snake*: The $f(s)$ curve is controlled by the locations of a smaller number of $\{x_k\}$ control points.

association between constraints (cities) and curve (tour) points to evolve over time (Szeliski 1989). Note that this is a soft variant of the popular *iterated closest point* data constraint that is often used in fitting or aligning surfaces to data points or to each other §11.4.1 (Besl and McKay 1992, Zhang 1994).

To compute a good solution to the TSP, the slippery spring data association energy is combined with a regular first order internal smoothness energy (4.35) to define the cost of a tour. The tour $f(s)$ is initialized as a small circle around the mean of the city points and σ is progressively lowered (Figure 4.54). For large σ values, the tour tries to stay near the centroid of the points, but as σ decreases, each city pulls more and more strongly on its closest tour points (Durbin *et al.* 1989). In the limit as $\sigma \rightarrow 0$, each city is guaranteed to capture at least one tour point and the tours between subsequent cities become straight lines.

Splines and shape priors

While snakes can be very good at capturing the fine and irregular detail in many real-world contours, they sometimes exhibit too many degrees of freedom, making it more likely that they can get trapped in local minima during their evolution.

One solution to this problem is to control the snake with fewer degrees of freedom through the use of B-spline approximations (Menet *et al.* 1990b, Menet *et al.* 1990a, Cipolla and Blake 1990). The resulting *B-snake* (Figure 4.55) can be written as

$$\mathbf{f}(s) = \sum_k B_k(s) \mathbf{x}_k \quad (4.42)$$

or in discrete form as

$$\mathbf{F} = \mathbf{B} \mathbf{X} \quad (4.43)$$

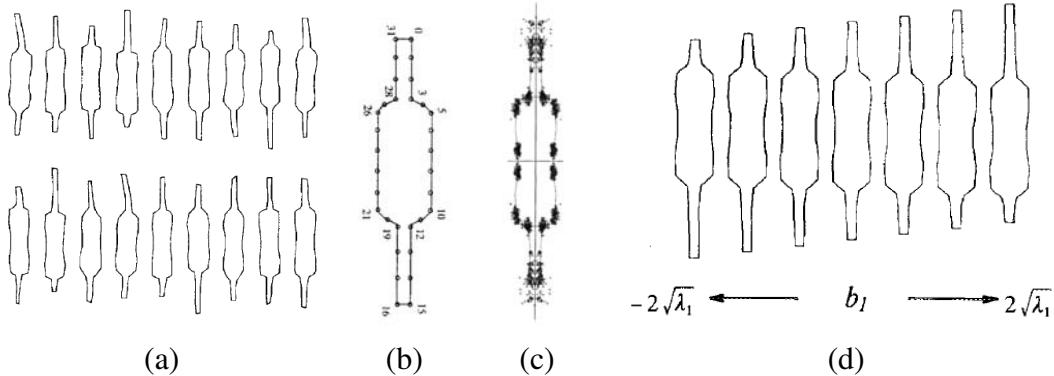


Figure 4.56: *Point distribution model for a set of resistors (Cootes et al. 1995)*: (a) set of input resistor shapes; (b) assignment of control points to the boundary; (c) distribution (scatter plot) of point locations; (d) first (largest) mode of variation in the ensemble shapes.

with

$$\mathbf{F} = \begin{bmatrix} \mathbf{f}^T(0) \\ \vdots \\ \mathbf{f}^T(N) \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} B_0(s_0) & \dots & B_K(s_0) \\ \vdots & \ddots & \vdots \\ B_0(s_N) & \dots & B_K(s_N) \end{bmatrix}, \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}^T(0) \\ \vdots \\ \mathbf{x}^T(K) \end{bmatrix}. \quad (4.44)$$

If the object being tracked or recognized has large variations in location, scale, or orientation, these can be modeled as an additional transformation on the control points, e.g., $\mathbf{x}'_k = s\mathbf{R}\mathbf{x}_k + \mathbf{t}$ (2.18), which can be estimated at the same time as the values of the control points. Alternatively, a separate *detection* and *alignment* stages can be run to first localize and orient the objects of interest (Cootes et al. 1995).

In a B-snake, because the snake is controlled by fewer degrees of freedom, there is less need for the internal smoothness forces used with the original snakes, although these can still be derived and implemented using finite element analysis, i.e., taking derivatives and integrals of the B-spline basis functions (Terzopoulos 1983, Bathe 2007).

In practice, it is more common to estimate a set of *shape priors* on the typical distribution of the control points $\{\mathbf{x}_k\}$ (Cootes et al. 1995). Consider the set of resistor shapes shown in Figure 4.56a. If we describe each contour with the set of control points shown in Figure 4.56b, we can plot the distribution of each point in a scatter plot, as in Figure 4.56c.

One potential way of describing this distribution would be by the location $\bar{\mathbf{x}}_k$ and 2-D covariance \mathbf{C}_k of each individual point \mathbf{x}_k . These could then be turned into a quadratic penalty (prior energy) on the point location,

$$E_{\text{loc}}(\mathbf{x}_k) = \frac{1}{2}(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T \mathbf{C}_k^{-1} (\mathbf{x}_k - \bar{\mathbf{x}}_k). \quad (4.45)$$

In practice, however, the variation in point locations is usually highly correlated.

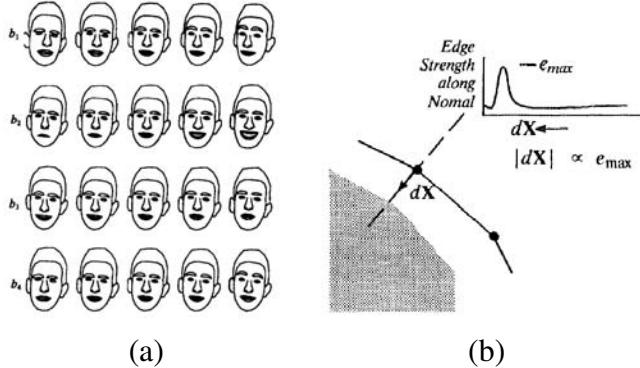


Figure 4.57: *Active Shape Model (ASM)*: (a) the effect of varying the first four shape parameters for a set of faces (Cootes et al. 1993); (b) searching for the strongest gradient along the normal to each control point (Cootes et al. 1995).

A preferable approach is to estimate the joint covariance of all the points simultaneously. First, let us concatenate all of the point locations $\{\mathbf{x}_k\}$ into a single vector \mathbf{x} , e.g., by interleaving the x and y locations of each point. The distribution of these vectors across all training examples (Figure 4.56a) can be described with a mean $\bar{\mathbf{x}}$ and a covariance

$$\mathbf{C} = \frac{1}{P} \sum_p (\mathbf{x}_p - \bar{\mathbf{x}})(\mathbf{x}_p - \bar{\mathbf{x}})^T, \quad (4.46)$$

where \mathbf{x}_p are the P training examples. Using *eigenvalue analysis* §B.1, which is also known as *Principal Component Analysis* (PCA), the covariance matrix can be written as (B.7),

$$\mathbf{C} = \Phi \operatorname{diag}(\lambda_0 \dots \lambda_{K-1}) \Phi^T. \quad (4.47)$$

In most cases, the likely appearance of the points can be modeled using only a few of eigenvectors with the largest eigenvalues. The resulting *point distribution model* (Cootes et al. 1993, Cootes et al. 1995) can be written as

$$\mathbf{x} = \bar{\mathbf{x}} + \hat{\Phi} \mathbf{b}, \quad (4.48)$$

where \mathbf{b} is an $M \ll K$ element *shape parameter* vector and $\hat{\Phi}$ are the first m columns of Φ . To constrain the shape parameters to reasonable values, a quadratic penalty of the form

$$E_{\text{shape}} = \frac{1}{2} \mathbf{b}^T \operatorname{diag}(\lambda_0 \dots \lambda_{M-1}) \mathbf{b} = \sum_m b_m^2 / 2\lambda_m. \quad (4.49)$$

Alternatively, the range of allowable b_m values can be limited to a range of values, e.g., $|b_m| \leq 3\sqrt{\lambda_m}$ (Cootes et al. 1995). Alternative approaches for deriving a set of shape vectors are reviewed in (Isard and Blake 1998).

Varying the individual shape parameters b_m over the range $-2\sqrt{\lambda_m} \leq b_m \leq 2\sqrt{\lambda_m}$ can give a good indication of the expected variation in appearance, as shown in Figure 4.56d. Another example, this time related to face contours, is shown in Figure 4.57a.

In order to align a point distribution model with an image, each control point searches in a direction normal to the contour to find the most likely corresponding image edge point (Figure 4.57b). These individual measurements can be combined with priors on the shape parameters (and, if desired, position, scale, and orientation parameters) to estimate a new set of parameters. The resulting *Active Shape Model* (ASM) can be iteratively minimized to fit images to non-rigidly deforming objects such as medical images and body parts such as hands (Cootes *et al.* 1995). The ASM can also be combined with a PCA analysis of the underlying gray-level distribution to create an *Active Appearance Model* (AAM) (Cootes *et al.* 2001), which is discussed in more detail in §14.1.2 .

DYNAMIC SNAKES AND CONDENSATION

In many applications of active contours, the object of interest is being tracked from frame to frame as it deforms and evolves. In this case, it makes sense to use estimates from the previous frame to predict and constrain the new estimates.

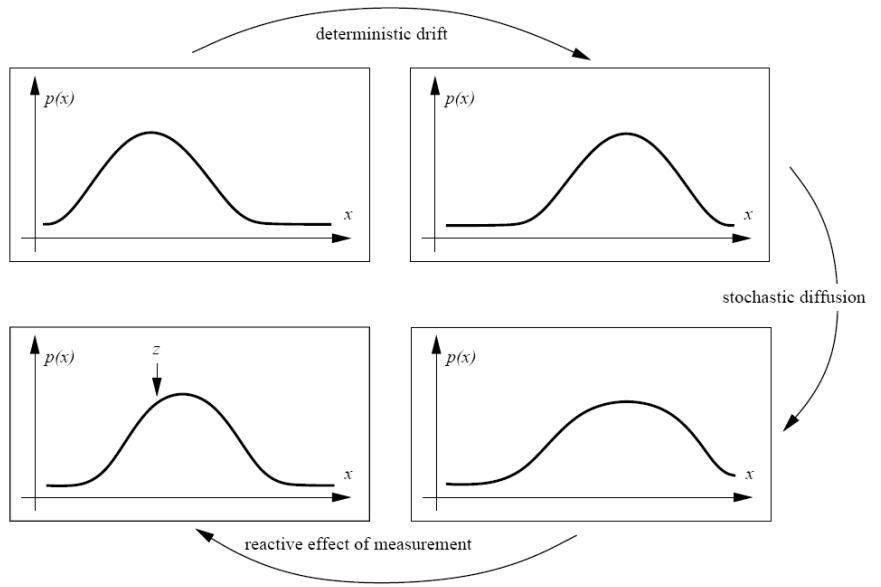
One way to do this is to use Kalman filtering (Appendix §B.8), which results in a formulation called *Kalman snakes* (Terzopoulos and Szeliski 1992, Blake *et al.* 1993). The Kalman filter is based on a linear dynamic model of shape parameter evolution,

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{w}, \quad (4.50)$$

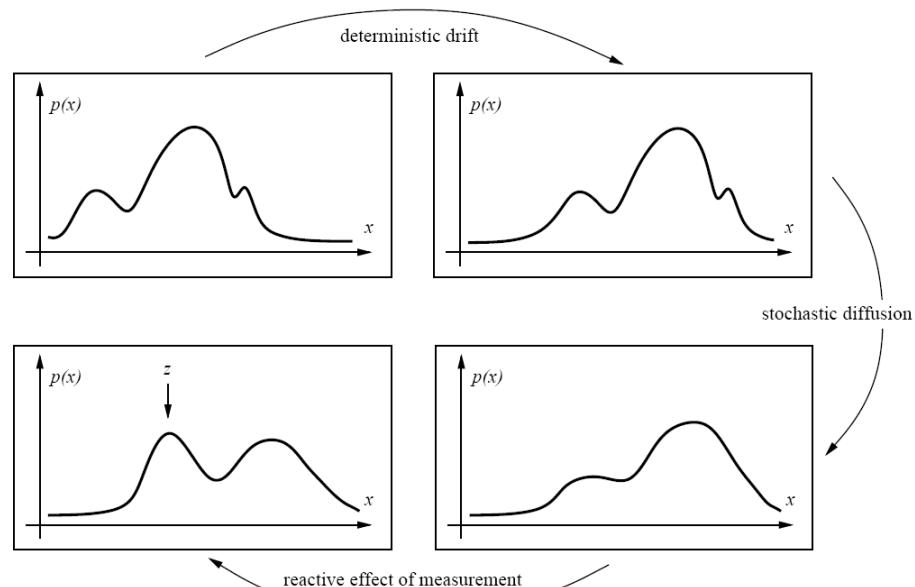
where \mathbf{x}_t and \mathbf{x}_{t-1} are the current state variables, \mathbf{A} is the linear *transition matrix*, and \mathbf{w} is a noise (perturbation) vector, which is often modeled as a Gaussian. The matrices \mathbf{A} and the noise covariance can be learned ahead of time by observing typical sequences of the object being tracked (Blake and Isard 1998).

While the exact formulas for the Kalman filter are given in Appendix §B.8, its qualitative behavior can be seen in Figure 4.58a. The linear dynamic model causes a deterministic change (drift) in the previous estimate, while the process noise (perturbation) causes a stochastic diffusion that increases the system entropy (lack of certainty). New measurement from the current frame restore some of the certainty (peakedness) in the updated estimate.

In many situations, however, such as when tracking in clutter, a better estimate for the contour can be obtained if we remove the assumptions that the distribution are Gaussian, which is what the Kalman filter requires. In this case, a general multi-modal distribution gets propagated, as shown in Figure 4.58b. In order to model such multi-modal distributions, Isard and Blake (1998) proposed the use of *particle filtering*, which represents the distribution as a weighted sum of point



(a)



(b)

Figure 4.58: *Probability density propagation (Isard and Blake 1998).* At the beginning of each estimation step, the probability density is updated according to the linear dynamic model (deterministic drift) and its certainty is reduced due to process noise (stochastic diffusion). New measurements introduce additional information that helps refine the current estimate. While the Kalman filter models the distributions as uni-modal, i.e., using a mean and covariance (a), some applications require more general multi-modal distributions (b).

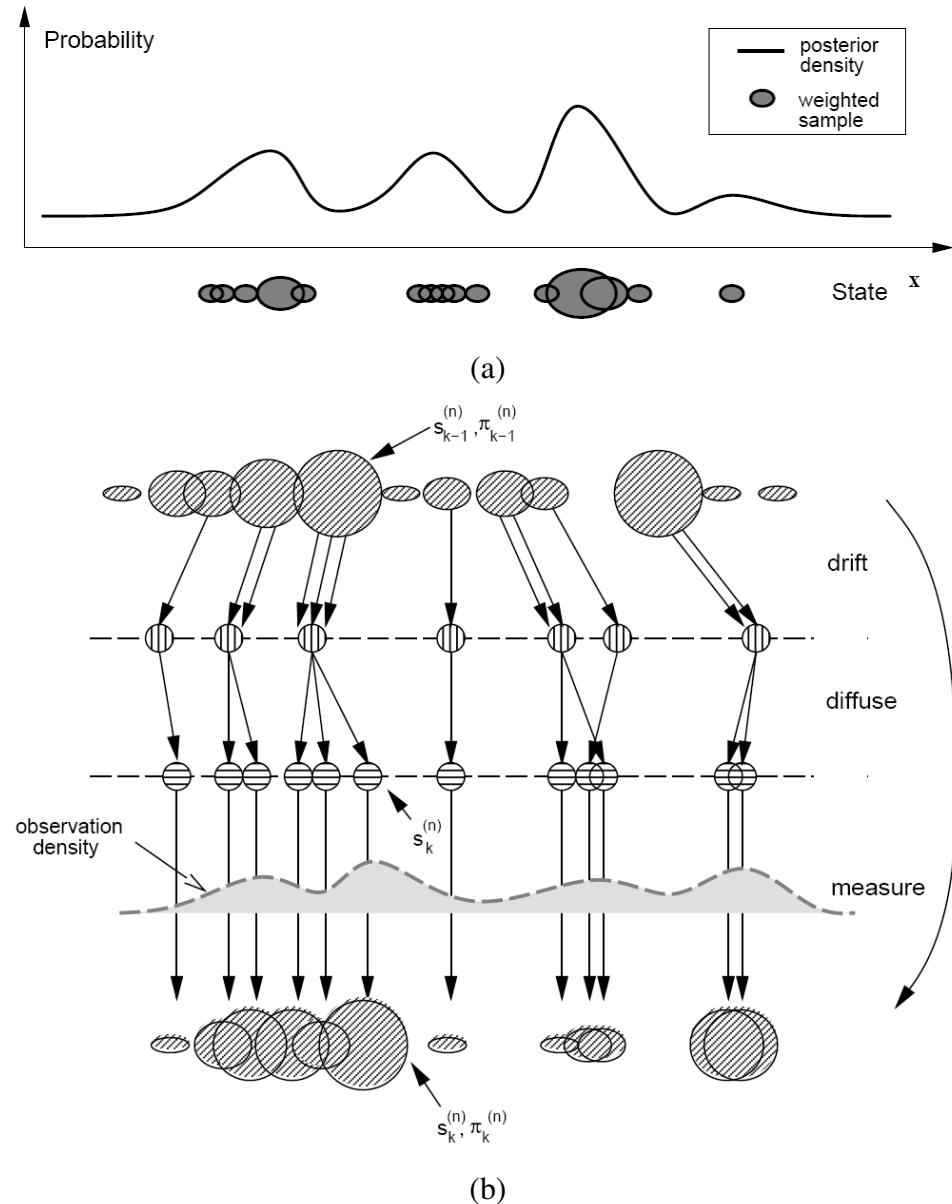


Figure 4.59: *Factored sampling using particle filter in the CONDENSATION algorithm (Isard and Blake 1998): (a) each density distribution is represented using a superposition of weighted particles; (b) the drift-diffusion-measurement cycle implemented using random sampling, perturbation, and re-weighting stages.*

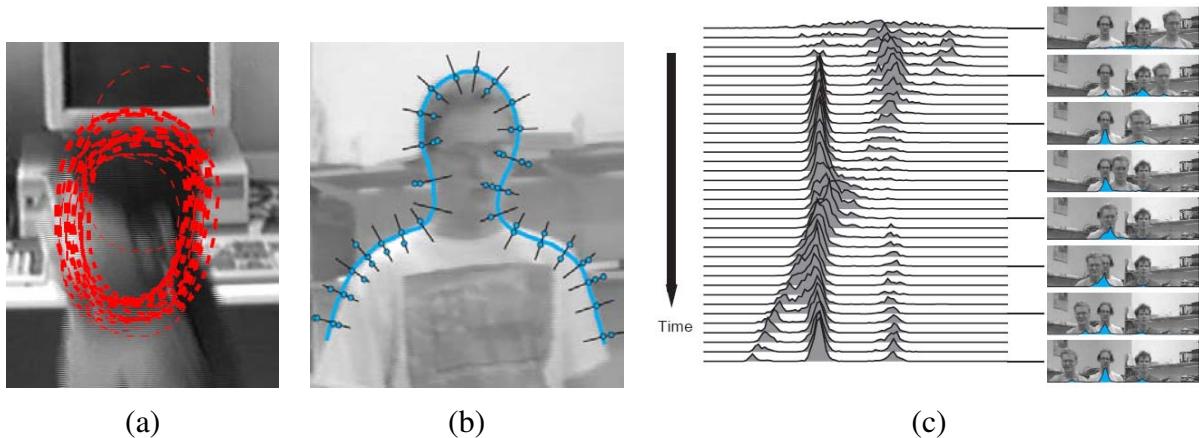


Figure 4.60: *Head tracking using CONDENSATION*: (Isard and Blake 1998): (a) sample set representation of head estimate distribution; (b) multiple measurements at each control vertex location; (c) multi-hypothesis tracking over time.

estimates, which is a variant on a mixture of Gaussians (Bishop 2006).⁹

Particle filtering (*aka factored sampling*) techniques represent a probability distribution using a collection of weighted point samples (Figure 4.59a). To update the locations of the samples according to the linear dynamics (deterministic drift), the centers of the samples are updated according to (4.50) and multiple samples are generated for each point (Figure 4.59b). These are then perturbed to account for the stochastic diffusion, i.e., their locations are moved by random vectors taken from the distribution of w .¹⁰ Finally, the weights of these samples are multiplied by the measurement probability density, i.e., we take each sample and measure its likelihood given the current (new) measurements. The exact formulas for these steps are given in Appendix §B.9. Because the point samples represent and propagate conditional estimates of the multi-modal density, Isard and Blake (1998) dubbed their algorithm CONDENSATION or CONDENSATION.

Figure 4.60a shows what a factored sample of a head tracker might look like, drawing a red B-spline contour for each of (a subset of) the particles being tracked. Figure 4.60b shows why the measurement density itself is often multi-modal: the locations of the edges perpendicular to the spline curve can have multiple local maxima due to background clutter. Finally, Figure 4.60c shows the temporal evolution of the the conditional density (x coordinate of the head and shoulder tracker centroid) as it tracks several people over time.

⁹ It is also related to *multiple-hypothesis tracking* techniques (Bar-Shalom and Fortmann 1988, Cham and Rehg 1999).

¹⁰ Note that because of the structure of these steps, non-linear dynamics and non-Gaussian noise can be used.

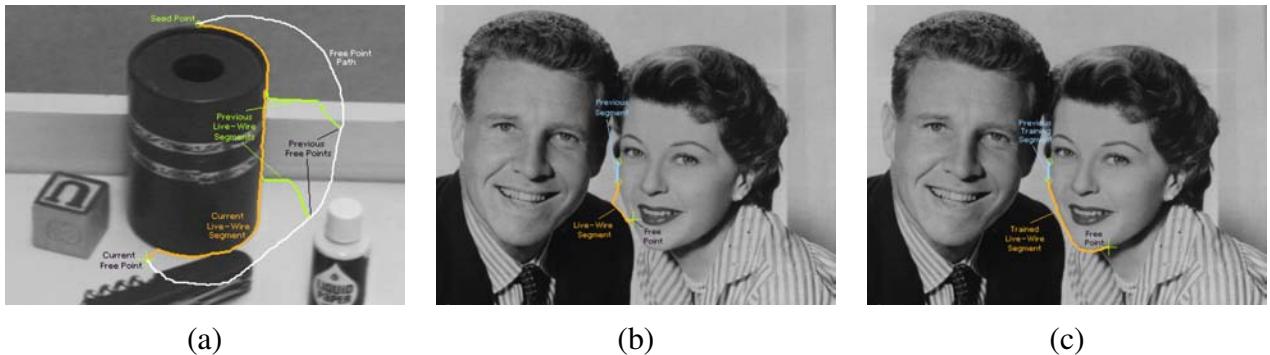


Figure 4.61: *Intelligent scissors* (Mortensen and Barrett 1995): (a) as the mouse traces the white path, the scissors follow the orange path along the object boundary (the green curves show intermediate curve positions); (b) regular scissors can sometimes jump to a stronger (incorrect) boundary; (c) after training to the previous segment, similar edge profiles are preferred.

4.4.2 Scissors

Active contours allow a user to roughly specify a boundary of interest and have the system evolve the contour towards a more accurate location as well as track it over time. The results of this curve evolution, however, may be unpredictable and may require additional user-based hints to achieve the desired result.

An alternative approach is to have the system optimize the contour in real time as the user is drawing (Mortensen 1999). The *intelligent scissors* system developed by Mortensen and Barrett (1995) does just that. As the user draws a rough outline (the white curve in Figure 4.61a), the system computes and draws a better curve that clings to high-contrast edges (orange curve).

To compute the optimal curve path (*live wire*), the image is first pre-processed to associate low costs with edges (links between neighboring horizontal, vertical, and diagonal, i.e., \mathcal{N}_8 neighbors) that are likely to be boundary elements. Their system uses a combination of zero-crossing, gradient magnitudes, and gradient orientations to compute these costs.

Next, as the user traces a rough curve, the system continuously recomputes the lowest-cost path between the starting *seed point* and the current mouse location using Dijkstra's algorithm, a breadth-first dynamic programming algorithm that terminates at the current target location.

In order to keep the system from jumping around unpredictably, the system will “freeze” the curve to date (reset the seed point) after a period of inactivity. To prevent the live wire from jumping onto adjacent higher-contrast contours, the system also “learns” the intensity profile under the current optimized curve, and uses this to preferentially keep the wire moving along the same (or a similar looking) boundary (Figure 4.61b–c).

Several extensions have been proposed to the basic algorithm, which works remarkably well even in its original form. Mortensen and Barrett (1999) use *tobogganing*, which is a simple form of

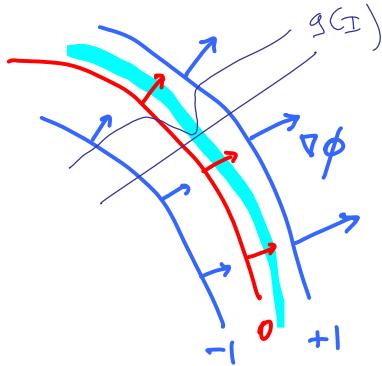


Figure 4.62: Level set evolution for a geodesic active contour. The embedding function ϕ is updated based on the curvature of the underlying surface modulated by the edge/speed function $g(I)$, as well as the gradient of $g(I)$, thereby attracting it so strong edges.

watershed region segmentation, to pre-segment the image into regions whose boundaries become candidates for optimized curve paths. The resulting region boundaries are turned into a much smaller graph, where nodes are located wherever three or four regions meet. The Dijkstra algorithm is then run on this reduced graph, resulting in much faster (and often more stable) performance. Another extension to intelligent scissors approach is to use a probabilistic framework that takes into account the current trajectory of the boundary, resulting in a system called JetStream (Pérez *et al.* 2001).

Instead of re-computing an optimal curve at each time instant, a simpler system can be developed by simply “snapping” the current mouse position to the nearest likely boundary point (Gleicher 1995). Applications of these boundary extraction techniques to image cutting and pasting are presented in §9.4.

4.4.3 Level Sets

A limitation of active contours based on parametric curves of the form $f(s)$, e.g., snakes, B-snakes, and CONDENSATION, is that it is difficult to change the topology of the curve as it evolves (McInerney and Terzopoulos 1993). Furthermore, if the shape changes dramatically, curve-reparametrization may also be required.

An alternative representation for such closed contours is to use a *level set*, where the zero-crossing(s) of a *characteristic* (or signed distance §3.2.4) function define the curve. Level sets evolve to fit and track objects of interest by modifying the underlying *embedding function* (another name for this 2D function) $\phi(x, y)$ instead of the curve $f(s)$ (Sethian 1999, Sapiro 2001, Osher and Paragios 2003). To reduce the amount of computation required, only a small strip (frontier) around the locations of the current zero-crossing needs to be updated at each step, which results in

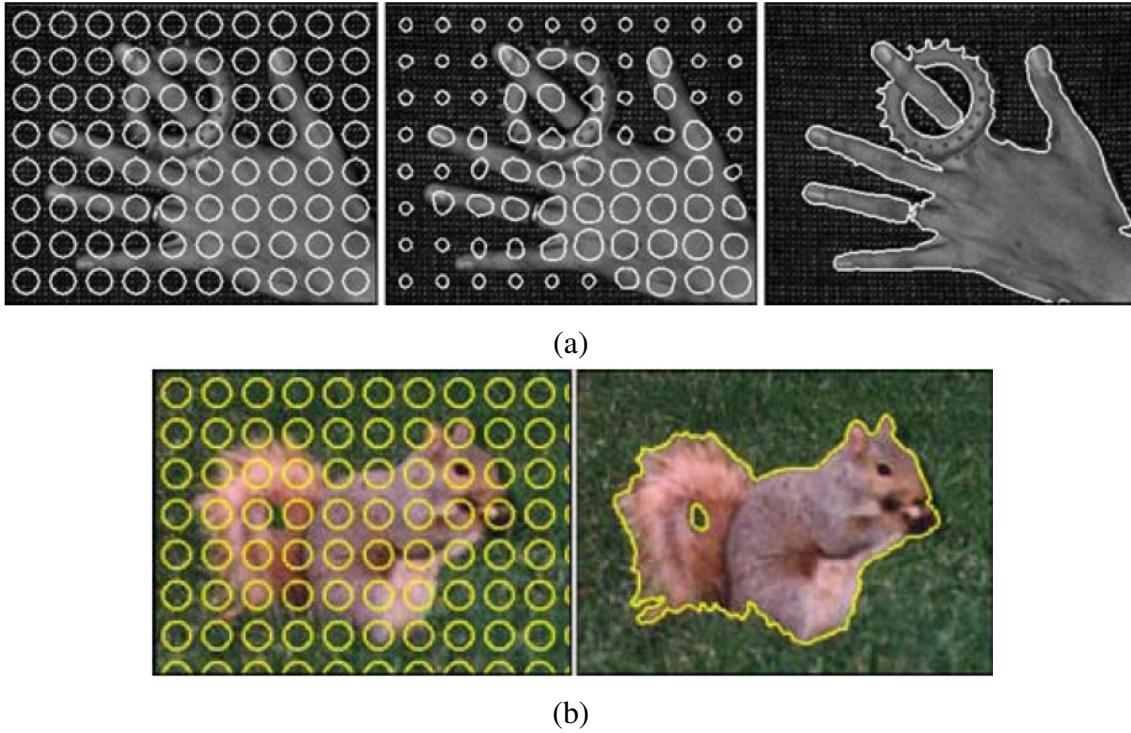


Figure 4.63: *Level set segmentation* (Cremers et al. 2007): (a) grayscale image segmentation and (b) color image segmentation. Uni-variate and multi-variate Gaussians are used to model the foreground and background pixel distributions. The initial circles evolve towards an accurate foreground/background segmentation, adapting their topology as they evolve.

what are called *fast marching methods* (Sethian 1999).

An example of an evolution equation is the *geodesic active contour* proposed by Caselles et al. (1997) and Yezzi et al. (1997),

$$\begin{aligned} \frac{d\phi}{dt} &= |\nabla\phi| \operatorname{div} \left(g(I) \frac{\nabla\phi}{|\nabla\phi|} \right) \\ &= g(I) |\nabla\phi| \operatorname{div} \left(\frac{\nabla\phi}{|\nabla\phi|} \right) + \nabla g(I) \cdot \nabla\phi, \end{aligned} \quad (4.51)$$

where $g(I)$ is a generalized version of the snake edge potential (4.37). To get an intuitive sense of the curve's behavior, assume that the embedding function ϕ is a signed distance function away from the curve (Figure 4.62), in which case $|\phi| = 1$. The first term in (4.51) moves the curve in the direction of its curvature, i.e., it acts to straighten the curve, under the influence of the modulation function $g(I)$. The second term moves the curve down the gradient of $g(I)$, encouraging the curve to migrate towards minima of $g(I)$.

While this level-set formulation can readily change topology, it is still susceptible to local

minima, since it is based on local measurements such as image gradients. An alternative approach is to re-cast the problem in a segmentation framework, where the energy measures the consistency of the image statistics (color, texture, motion) inside and outside the segmented regions (Cremers *et al.* 2007, Rousson and Paragios 2008, Houhou *et al.* 2008). These approaches build on earlier energy-based segmentation frameworks introduced by Leclerc (1989), Mumford and Shah (1989), and Chan and Vese (1992), which are discussed in more detail in §4.5.4. Examples of such level-set segmentations are shown in Figure 4.63, which show the evolution of the level sets from a series of distributed circles towards the final binary foreground/background segmentation.

[Note: Mention workshops devoted to this topic?]

4.4.4 Application: Contour tracking and rotoscoping

Active contours can be used in a wide variety of object tracking applications (Blake and Isard 1998). For example, they can be used to track facial features for performance-driven animation (Terzopoulos and Waters 1990, Lee *et al.* 1995, Parke and Waters 1996, Bregler *et al.* 1997) (Figure 4.53b). They can also be used to perform head and person tracking, as shown in Figure 4.60, as well as moving vehicle tracking [Note: find some reference, probably level-set literature]. Additional applications include medical image segmentation, where contours can either be tracked from slice to slice in computerized tomography (3D medical imagery) (Cootes and Taylor 2001), or over time, as in ultrasound scans.

An interesting application that is closer to computer animation and visual effects is *rotoscoping*, which uses the tracked contours to deform a set of hand-drawn animations (or to modify or replace the original video frames).¹¹ Agarwala *et al.* (2004) present a system based on tracking hand-drawn B-spline contours drawn at selected keyframes, using a combination of geometric and appearance-based criteria (Figure 4.64). They also provide an excellent review of previous rotoscoping and image-based contour tracking systems.

Additional applications of rotoscoping (object contour detection and segmentation), such as cutting and pasting object from one photograph into another, are presented in §9.4.

4.5 Region segmentation

Image segmentation is the task of finding groups of pixels that “go together”. In statistics, this problem is known as *cluster analysis* and is a widely studied area with hundreds of different algorithms (Jain and Dubes 1988, Kaufman and Rousseeuw 1990, Jain *et al.* 2000, Jain *et al.* 2004).

¹¹ The term comes from the original rotoscope, which was a device that projected frames of a live-action film underneath an acetate so that artists could draw animations directly over the actors’ shapes.

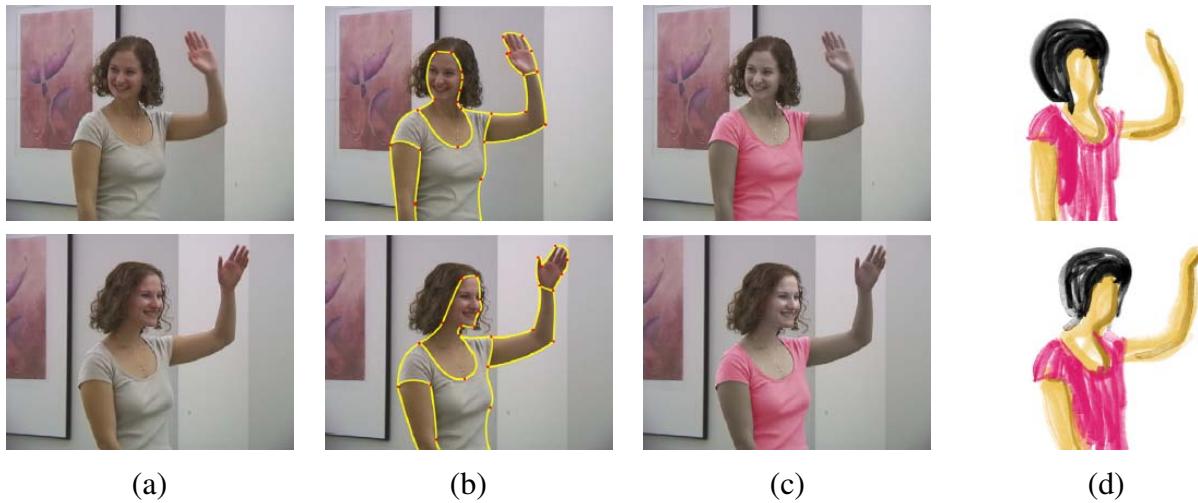


Figure 4.64: *Keyframe-based rotoscoping* (Agarwala et al. 2004): (a) original frame; (b) rotoscoped contours; (c) re-colored blouse; (d) rotoscoped hand-drawn animation.

In computer vision, image segmentation is one of the oldest and most widely studied problems (Brice and Fennema 1970, Pavlidis 1997, Riseman and Arbib 1977, Ohlander *et al.* 1978, Rosenfeld and Davis 1979, Haralick and Shapiro 1985). Early techniques tend to use region splitting or merging (Brice and Fennema 1970, Ohlander *et al.* 1978), which correspond to *divisive* and *agglomerative* algorithms in the clustering literature (Jain *et al.* 2004). More recent algorithms often optimize some global criterion, such as intra-region consistency and inter-region boundary lengths or dissimilarity (Leclerc 1989, Mumford and Shah 1989, Shi and Malik 2000, Felzenszwalb and Huttenlocher 2004a).

We have already seen examples of image segmentation in the sections on thresholding and connected components §3.2.3, binary Markov Random Fields §3.6.2, active contours §4.4, and level sets §4.4.3. In this section, I review some additional techniques that have been developed for image segmentation. These include algorithms based on regions splitting and merging §4.5.1, *mean shift* (mode finding) §4.5.2, *normalized cuts* (splitting based on pixel similarity metrics) §4.5.3, and binary Markov Random Fields solved using graph cuts §4.5.4.

Since the literature on image segmentation is so vast, a good way to get a handle on some of the better performing algorithms is to look at experimental comparisons on human-labelled databases. The best known of these is the The Berkeley Segmentation Dataset and Benchmark¹² (Martin *et al.* 2001), which consists of 300 images from a Corel image dataset that was hand-labelled by 30 human subjects. Many of the more recent image segmentation algorithms report comparative results on this database, and Unnikrishnan *et al.* (2007) propose new metrics for comparing such algorithms. A new database of foreground/background segmentations used in (Alpert *et al.* 2007)

¹² <http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/>

is also available.¹³

4.5.1 Split and merge

As mentioned before, the simplest possible technique for segmenting a grayscale image is to select a threshold and to then compute connected components §3.2.3. Unfortunately, a single threshold is rarely sufficient for the whole image because of lighting and intra-object statistical variations.

In this section, I describe a number of algorithms that proceed by either recursively splitting the whole image into pieces based on region statistics, or, conversely, merging pixels and regions together in a hierarchical fashion.

Watershed

A technique related to thresholding, since it operates on a grayscale image, is *watershed* computation (Vincent and Soille 1991). This technique segments an image into several *catchment basins*, which are the regions of an image (interpreted as a height field or landscape) where rain would flow into the same lake. An efficient way to compute such regions is to start flooding the landscape at all of the local minima and to label ridges wherever different evolving components meet. The whole algorithm can be implemented using a priority queue of pixels and breadth-first search (Vincent and Soille 1991).¹⁴

Since images rarely have dark regions separated by lighter ridges, watershed segmentation is usually applied to a smoothed version of the gradient magnitude image (which also makes it usable with color images). This ends up finding smooth regions separated by visible (higher gradient) boundaries. Since such boundaries are what active contours usually follow, precomputing such a segmentation using either watershed or the related *tobogganing* technique §4.4.2 is often used in active contour algorithms (Mortensen and Barrett 1999, Li *et al.* 2004b).

Unfortunately, watershed segmentation associates a unique region with each local minimum, which can lead to over-segmentation. Watershed segmentation is therefore often used as part of an interactive system, where the user first marks seed locations (with a click or a short stroke) that correspond to the centers of different desired components. Figure 4.65 shows the results of running the watershed algorithm with some manually placed markers on a confocal microscopy image. It also shows the result for an improved version of watershed that uses local morphology to smooth out and optimize the boundaries separating the regions (Beare 2006).

¹³ http://www.wisdom.weizmann.ac.il/~vision/Seg_Evaluation_DB/index.html

¹⁴ A related algorithm can be used to efficiently compute Maximally Stable Extremal Regions (MSERs) §4.1.1 (Nistér and Stewénius 2008).

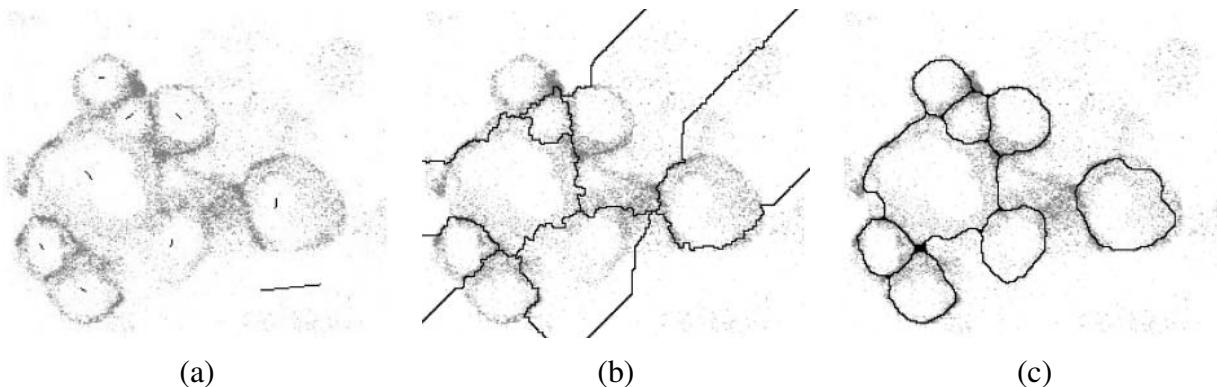


Figure 4.65: *Locally constrained watershed segmentation* (Beare 2006): (a) original confocal microscopy image with marked seeds (line segments); (b) standard watershed segmentation; (c) locally constrained watershed segmentation.

Region splitting (divisive clustering)

Splitting the image into successively finer regions is one of the oldest techniques in computer vision. Ohlander *et al.* (1978) present such a technique, which first computes a histogram for the whole image, and then finds a threshold that best separates the large peaks in the histogram. This process is repeated until regions are either fairly uniform or below a certain size.

More recent splitting algorithm often optimize some metric of intra-region similarity and inter-region dissimilarity. These are covered in the sections on normalized cuts §4.5.3 and graph cuts §4.5.4.

Region merging (agglomerative clustering)

Region merging techniques also date back to the beginnings of computer vision. Brice and Fenema (1970) use a dual grid for representing boundaries between pixels and merge regions based on their relative boundary lengths and the strength of the visible edges at these boundaries.

In data clustering, algorithms can link clusters together based either on the distance between their closest points (single-link clustering), their farthest points (complete-link clustering), or something in between (Jain *et al.* 2004). Kamvar *et al.* (2002) provide a probabilistic interpretation of these algorithms and show how additional models can be incorporated within this framework.

A very simple version of pixel-based merging is to simply merge adjacent regions whose average color difference is below a threshold or whose regions are too small. Segmenting the image into such *superpixels* (Mori *et al.* 2004), which are in themselves not semantically meaningful, can be a useful pre-processing stage to make higher-level algorithms such as stereo matching (Zitnick *et al.* 2004, Taguchi *et al.* 2008), optic flow (Zitnick *et al.* 2005), and recognition (Mori *et al.* 2004,

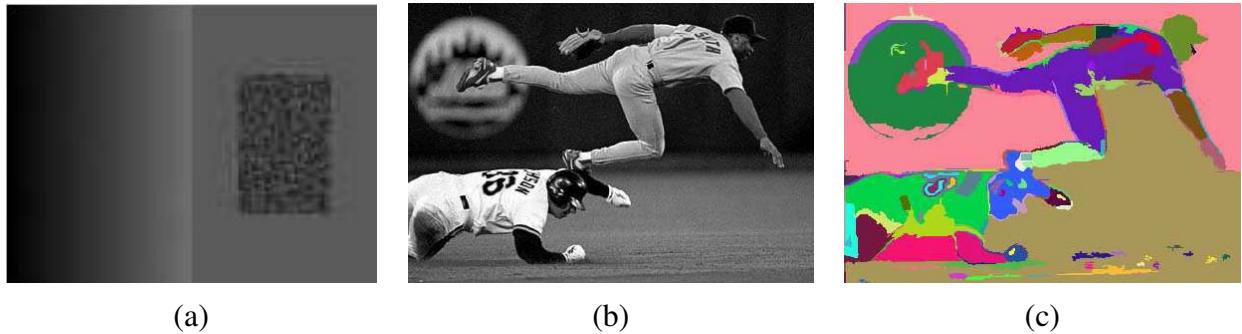


Figure 4.66: *Graph-based merging segmentation* ([Felzenszwalb and Huttenlocher 2004a](#)): (a) input grayscale image that is successfully segmented into three regions even though the variation inside the smaller rectangle is larger than the variation across the middle edge; (b) input grayscale image; (c) resulting segmentation using an \mathcal{N}_8 pixel neighborhood.

[Mori 2005](#)) both faster and more robust.

Graph-based segmentation

While many merging algorithms simply apply a fixed rule to group pixels and regions together, [Felzenszwalb and Huttenlocher \(2004a\)](#) present a merging algorithm that uses *relative dissimilarities* between regions to determine which ones should be merged, which produces an algorithm that provably optimizes a global grouping metric. They start with a pixel-to-pixel dissimilarity measure $w(e)$ such as intensity differences between \mathcal{N}_8 neighbors. (Alternatively, they also use the *joint feature space* distances (4.74) introduced by [Comaniciu and Meer \(2002\)](#), which I discuss in the next section §4.5.2.)

For any region R , its *internal difference* is defined as the largest edge weight in the region's minimum spanning tree,

$$\text{Int}(R) = \min_{e \in \text{MST}(R)} w(e). \quad (4.52)$$

For any two adjacent regions with at least one edge connecting their vertices, the difference between these regions is defined as the minimum weight edge connecting the two regions,

$$\text{Dif}(R_1, R_2) = \min_{e=(v_1, v_2) | v_1 \in R_1, v_2 \in R_2} w(e). \quad (4.53)$$

Their algorithm merges any two adjacent regions whose difference is smaller than the minimum internal difference of these two regions,

$$\text{MInt}(R_1, R_2) = \min(\text{Int}(R_1) + \tau(R_1), \text{Int}(R_2) + \tau(R_2)), \quad (4.54)$$

where $\tau(R)$ is a heuristic region penalty they set to $k/|R|$, but which can be set to any application-specific measure of region goodness.

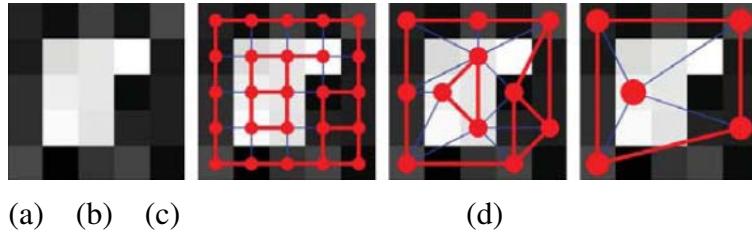


Figure 4.67: *Coarse to fine node aggregation in segmentation by weighted aggregation (SWA)* (Sharon et al. 2006): (a) original gray-level pixel grid; (b) inter-pixel couplings, where thicker lines indicate stronger couplings; (c) after one level of coarsening, where each original pixel is strongly coupled to one of the coarse-level nodes; (d) after two levels of coarsening.

By merging regions in decreasing order of the edges separating them (which can be efficiently evaluated using a variant of Kruskal’s minimum spanning tree algorithm), they provably produce segmentations that are neither too fine (there exist regions that could have been merged) nor too coarse (there are regions that could be split without being mergeable). For fixed-sized pixel neighborhoods, the running time for this algorithm is $O(n \log n)$, where n is the number of image pixels, which makes it among the fastest segmentation algorithms (Paris and Durand 2007). Figure 4.66 shows two examples of images segmented using their technique.

Probabilistic aggregation

Alpert et al. (2007) develop a probabilistic merging algorithm based on two cues, namely gray-level similarity and texture similarity. The gray-level similarity between regions R_i and R_j is based on the *minimal external difference* to other neighboring regions,

$$\sigma_{local}^+ = \min(\Delta_i^+, \Delta_j^+), \quad (4.55)$$

where $\Delta_i^+ = \min_k |\Delta_{ik}|$ and Δ_{ik} is the difference in average intensities between regions R_i and R_k . This gets compared to the *average intensity difference*,

$$\sigma_{local}^- = \frac{\Delta_i^- + \Delta_j^-}{2}, \quad (4.56)$$

where $\Delta_i^- = \sum_k (\tau_{ik} \Delta_{ik}) / \sum_k (\tau_{ik})$ and τ_{ik} is the boundary length between regions R_i and R_k . The texture similarity is defined using relative differences between histogram bins of simple oriented Sobel filter responses. The pairwise statistics σ_{local}^+ and σ_{local}^- are used to compute the likelihoods p_{ij} that two regions should be merged. (See their paper for more details.)

Merging proceeds in a hierarchical fashion inspired by algebraic multigrid techniques (Brandt 1986, Briggs et al. 2000) and previously used by the authors in their segmentation by weighted aggregation (SWA) algorithm (Sharon et al. 2006), which I discuss in §4.5.3. A subset of the

nodes $C \subset V$ that are (collectively) *strongly coupled* to all of the original nodes (regions) are used to define the problem at a coarser scale (Figure 4.67), where strong coupling is defined as

$$\frac{\sum_{j \in C} p_{ij}}{\sum_{j \in V} p_{ij}} > \phi, \quad (4.57)$$

with ϕ usually set to 0.2. The intensity and texture similarity statistics for the coarser nodes are recursively computed using weighted averaging, where the relative strengths (couplings) between coarse and fine level nodes are based on their merge probabilities p_{ij} . This allows the algorithm to run in essentially $O(n)$ time, using the same kind of hierarchical aggregation operations that are used in pyramid-based filtering or preconditioning algorithms. After a segmentation has been identified at a coarser level, the exact memberships of each pixel are computed by propagating coarse-level assignments to their finer-level “children” (Sharon *et al.* 2006, Alpert *et al.* 2007).

Figure 4.75 (later on in this section) shows the segmentations produced by this algorithm compared to a number of other popular segmentation algorithms. According to the authors, their probabilistic bottom-up merging algorithm outperforms previously developed segmentation algorithms, including their own SWA algorithm.

4.5.2 Mean shift and mode finding

Mean shift and mode finding techniques such as k-means and mixtures of Gaussians model the feature vectors associated with each pixel (e.g., color and position) as samples from an unknown probability density function and then try to find clusters (modes) in this distribution.

Consider the color image shown in Figure 4.68a. How would you segment this image based on color alone? Figure 4.68b shows the distribution of pixels in $L^*u^*v^*$ space, which is equivalent to what a vision algorithm that ignores spatial location would see. To make the visualization simpler, let’s only consider the L^*u^* coordinates, as shown in Figure 4.68c. How many obvious (elongated) clusters do you see? How would you go about finding these clusters?

The *k-means* and *mixtures of Gaussian* techniques use a *parametric* model of the density function to answer this question, i.e., they assume the density is the superposition of a small number of simpler distributions (e.g., Gaussians) whose locations (centers) and shape (covariance) can be estimated. Mean shift, on the other hand, smooths the distribution and finds its peaks as well as the regions of feature space that correspond to each peak. Since a complete density is being modeled, this approach is called *non-parametric* (Bishop 2006). Let us look at these techniques in more detail.

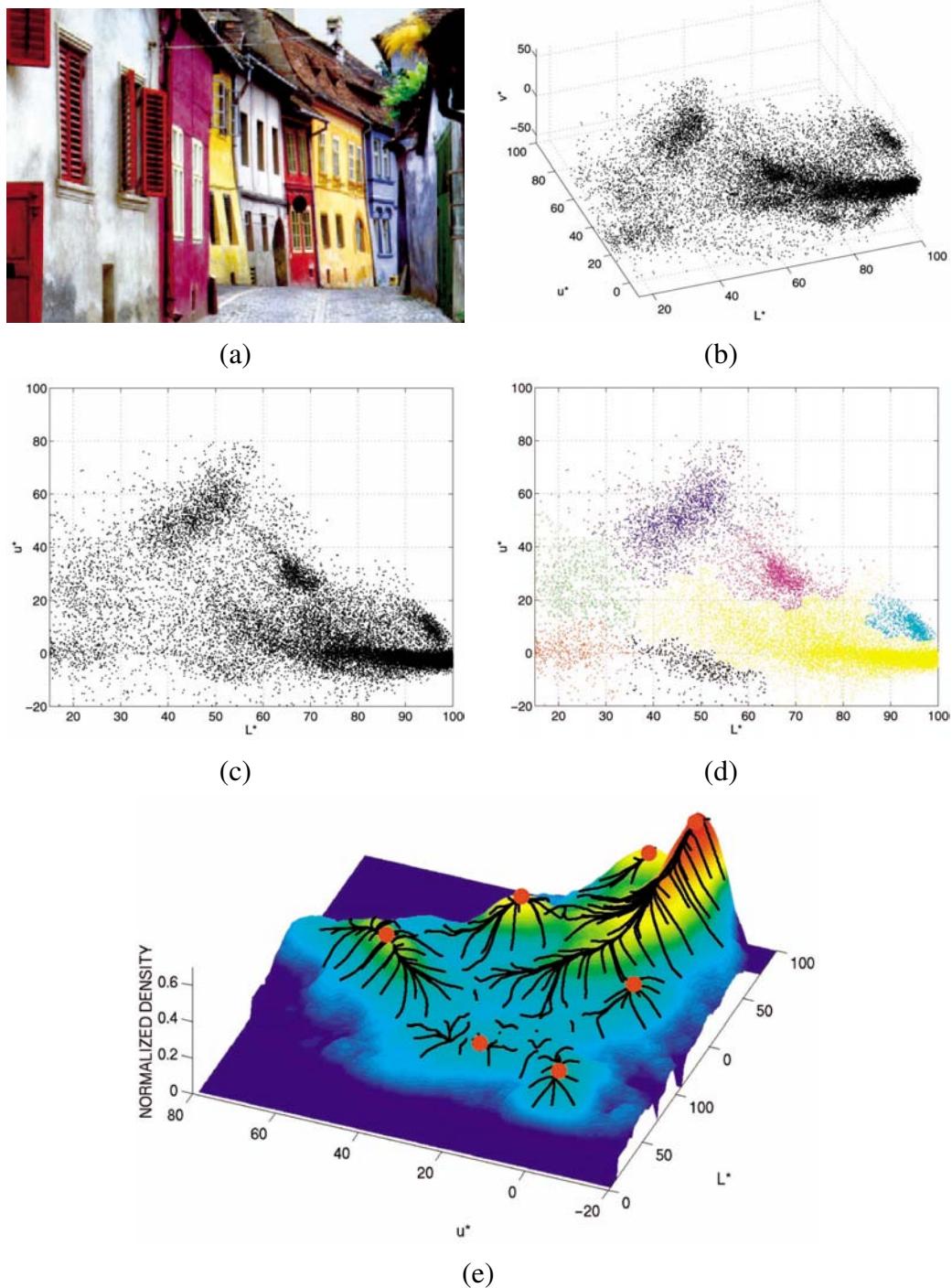
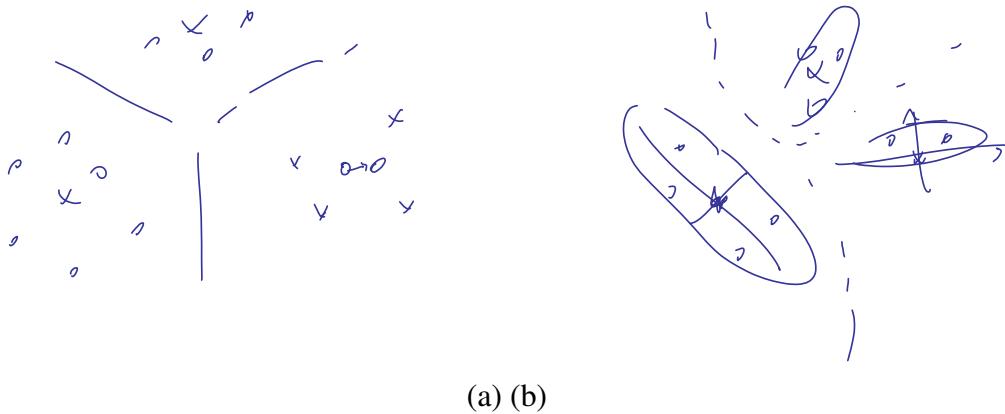


Figure 4.68: *Mean shift image segmentation* (Comaniciu and Meer 2002): (a) input color image; (b) pixels plotted in $L^*u^*v^*$ space; (c) L^*u^* space distribution; (d) clustered results after 159 mean shift procedures; (e) corresponding trajectories with peaks marked as red dots.



(a) (b)

Figure 4.69: *K-means and mixtures of Gaussians*. (a) In *k-means*, each sample is associated with its nearest cluster center. (The tessellation of the feature space is a Voronoi diagram and is indicated with dashed lines.) Cluster centers get updated to the mean of their constituent samples. (b) In *mixture of Gaussian modeling*, each cluster center is augmented with a covariance matrix (shown as an ellipse). Membership boundaries are no longer piecewise planar, and covariance estimates get updated along with cluster centers.

[Note: Generate two proper figures and fix (a) (b) labels]

K-means and Gaussian mixtures

While k-means implicitly models the probability density as a superposition of spherically symmetric distributions, in practice, it does not require any probabilistic reasoning or modeling (Bishop 2006). Instead, the algorithm is given the number of clusters k it is supposed to find, and it then iteratively updates the cluster center location based on the samples that are closest to each center (Figure 4.69a). The algorithm can be initialized by randomly sampling k centers from the input feature vectors. Techniques have also been developed for splitting or merging cluster centers based on their statistics, and for accelerating the process of finding the nearest mean center (Bishop 2006).

In mixtures of Gaussians, each cluster center is augmented by a covariance matrix whose values are re-estimated from the corresponding samples. Instead of using nearest neighbors to associate input samples with cluster centers, a *Mahalanobis distance*

$$d(\mathbf{x}_i, \boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k) = \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k^{-1}} = (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \quad (4.58)$$

is used, where \mathbf{x}_i are the input samples, $\boldsymbol{\mu}_k$ are the cluster centers, and $\boldsymbol{\Sigma}_k$ are their covariance estimates (Figure 4.69b). Samples can either be associated with the nearest cluster center (a *hard assignment* of membership), or they can be *softly assigned* to several nearby clusters.

[Note: Should the following description of mixtures of Gaussian and EM be moved to a section of Appendix B?]

This latter, more commonly used, approach corresponds to iteratively re-estimating the parameters for a mixture of Gaussians density function

$$p(\mathbf{x}|\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (4.59)$$

where π_k are the *mixing coefficients*, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the Gaussian mean and covariances, and

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{|\boldsymbol{\Sigma}_k|} e^{-d(\mathbf{x}, \boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k)} \quad (4.60)$$

is the *normal* (Gaussian) distribution (Bishop 2006).

To iteratively compute (a local) maximum likely estimate for the unknown mixture parameters $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$, the *expectation maximization* (EM) algorithm (Dempster *et al.* 1977) proceeds in two alternating stages:

1. The *expectation* stage (E step) estimates the *responsibilities*

$$z_{ik} = \frac{1}{Z_i} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad \text{with} \quad Z_i = \sum_k z_{ik}, \quad (4.61)$$

which are the estimates of how likely a sample \mathbf{x}_i was generated from the k th Gaussian (cluster).

2. The *maximization* stage (M step) updates the parameter values

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_i z_{ik} \mathbf{x}_i, \quad (4.62)$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_i z_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T, \quad (4.63)$$

$$\pi_k = \frac{N_k}{N}, \quad (4.64)$$

where

$$N_k = \sum_i z_{ik}. \quad (4.65)$$

is an estimate of the number of sample points assigned to each cluster.

Bishop (2006) has a wonderful exposition of both mixture of Gaussians estimation and the more general topic of expectation maximization.

In the context of image segmentation, Ma *et al.* (2007) present a nice review of segmentation using mixtures of Gaussian and develop their own extension based on Minimum Description Length (MDL) coding, which they show produces good results on the Berkeley segmentation database.

Mean shift

While k-means and mixtures of Gaussian use a parametric form to model the probability density function being segmented, mean shift implicitly models this distribution using a smooth continuous *non-parametric model*. The key to mean shift is a technique for efficiently finding peaks in this high-dimensional data distribution without ever computing the complete function explicitly (Fukunaga and Hostetler 1975, Cheng 1995, Comaniciu and Meer 2002).

Consider once again the data points shown in Figure 4.68c, which can be thought of as having been drawn from some probability density function. If we could compute this density function, as visualized in Figure 4.68e, we could find its major peaks (*modes*) and identify regions of the input space that climb to the same peak as being part of the same region. (This is the inverse of a *watershed* algorithm described earlier, which climbs downhill to find *basins of attraction*.)

The first question, then, is how to estimate the density function given a sparse set of samples. One of the simplest approaches is to just smooth the data, e.g., by convolving it with a fixed kernel of width h ,

$$f(\mathbf{x}) = \sum_i K(\mathbf{x} - \mathbf{x}_i) = \sum_i k\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h^2}\right), \quad (4.66)$$

where \mathbf{x}_i are the input samples and $k(r)$ is the kernel function (or *Parzen window*).¹⁵ This approach is known as *kernel density estimation* or the *Parzen window technique* (Duda *et al.* 2001, §4.3)(Bishop 2006, §2.5.1). Once we have computed $f(\mathbf{x})$, as shown in Figures 4.68e and 4.70, we can find its local maxima using gradient descent or some other optimization technique.

The problem with this “brute force” approach is that for higher dimensions, it becomes computationally prohibitive to evaluate $f(\mathbf{x})$ over the complete search space.¹⁶ Instead, mean shift uses a variant of what is known as *multiple restart gradient descent* in the optimization literature. Starting at some guess for a local maximum, \mathbf{y}_k , which can be a random input data point \mathbf{x}_i , mean shift computes the gradient of the density estimate $f(\mathbf{x})$ at \mathbf{y}_k and takes an uphill step in that direction (Figure 4.70). The gradient of $f(\mathbf{x})$ is given by

$$\nabla f(\mathbf{x}) = \sum_i (\mathbf{x}_i - \mathbf{x}) G(\mathbf{x} - \mathbf{x}_i) = \sum_i (\mathbf{x}_i - \mathbf{x}) g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h^2}\right), \quad (4.67)$$

where

$$g(r) = -k'(r), \quad (4.68)$$

¹⁵ In this simplified formulat, a Euclidean metric is used. I discuss a little later (4.74) how to generalize this to non-uniform (scaled or oriented) metrics. Note also that this distribution may not be *proper*, i.e., integrate to 1. Since we are looking for maxima in the density, this does not matter.

¹⁶ Even for 1-D, if the space is extremely sparse, it may be inefficient.

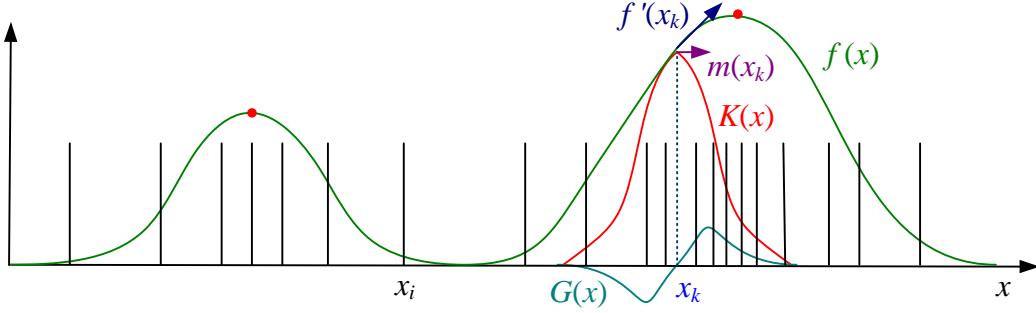


Figure 4.70: One dimensional visualization of the kernel density estimate, its derivative, and a mean shift. The kernel density estimate $f(x)$ is obtained by convolving the sparse set of input samples x_i with the kernel function $K(x) = k(x^2)$. The derivative of this function, $f'(x)$ can be obtained by convolving the inputs with the derivative kernel $G(x) = xg(x^2)$. Estimating the local displacement vectors around a current estimate x_k results in the mean shift vector $m(x_k)$, which in a multi-dimensional setting point in the same direction as the function gradient $\nabla f(\mathbf{x}_k)$. The red dots indicate local maxima in $f(x)$, which is where the mean shifts converge to.

and $k'(r)$ is the first derivative of $k(r)$. We can re-write the gradient of the density function as

$$\nabla f(\mathbf{x}) = \left[\sum_i G(\mathbf{x} - \mathbf{x}_i) \right] \mathbf{m}(\mathbf{x}), \quad (4.69)$$

where the vector

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_i \mathbf{x}_i G(\mathbf{x} - \mathbf{x}_i)}{\sum_i G(\mathbf{x} - \mathbf{x}_i)} - \mathbf{x} \quad (4.70)$$

is called the *mean shift*, since it is the difference between the weighted mean of the neighbors \mathbf{x}_i around \mathbf{x} and the current value of \mathbf{x} .

In the mean shift procedure, the current estimate of the mode \mathbf{y}_k at iteration k is replaced by its locally weighted mean,

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \mathbf{m}(\mathbf{y}_k) = \frac{\sum_i \mathbf{x}_i G(\mathbf{y}_k - \mathbf{x}_i)}{\sum_i G(\mathbf{y}_k - \mathbf{x}_i)}. \quad (4.71)$$

Comaniciu and Meer (2002) prove that this algorithm converges to a local maximum of $f(\mathbf{x})$ under reasonably weak conditions on the kernel $k(r)$, i.e., that it is monotonically decreasing. This convergence is not guaranteed for regular gradient descent unless appropriate step size control is used.

The two kernels that Comaniciu and Meer (2002) study are the Epanechnikov kernel,

$$k_E(r) = \max(0, 1 - r), \quad (4.72)$$

which is a radial generalization of a bilinear kernel, and the Gaussian (normal) kernel,

$$k_N(r) = \exp\left(-\frac{1}{2}r^2\right). \quad (4.73)$$

The corresponding derivative kernels $g(r)$ are a unit ball and another Gaussian, respectively. Using the Epanechnikov kernel converges in a finite number of steps, while the Gaussian kernel has a smoother trajectory (and produces better results) but converges very slowly near a mode (Exercise 4.22).

The simplest way to apply mean shift is to start a separate mean shift mode estimate \mathbf{y} at every input point \mathbf{x}_i and to iterate for a fixed number of steps or until the mean shift magnitude is below a threshold. A faster approach is to randomly subsample the input points \mathbf{x}_i and to find the nearest mode sample evolution path to the remaining points (Comaniciu and Meer 2002). Paris and Durand (2007) review a number of other more efficient implementations of mean shift, including their own approach, which is based on using an efficient low resolution estimate of the complete multi-dimensional space of $f(\mathbf{x})$ and finding its stationary points.

The color-based segmentation shown in Figure 4.68 only looks at pixel colors when determining the best clustering. It may therefore cluster together small isolated pixels that happen to have the same color, which may not correspond to a semantically meaningful segmentation of the image.

Better results can usually be obtained by clustering in the *joint domain* of color and location. In this approach, the spatial coordinates of the image $\mathbf{x}_s = (x, y)$, which are called the *spatial domain*, are concatenated with the color values \mathbf{x}_r , which are known as the *range domain*, and mean shift clustering is applied in this five-dimensional space \mathbf{x}_j . Since location and color may have different scales, the kernels are adjusted accordingly, i.e., we use a kernel of the form

$$K(\mathbf{x}_j) = k\left(\frac{\|\mathbf{x}_r\|^2}{h_r^2}\right) k\left(\frac{\|\mathbf{x}_s\|^2}{h_s^2}\right), \quad (4.74)$$

where separate parameters h_s and h_r are used to control the spatial and range bandwidths of the filter kernels. Figure 4.71 shows an example of mean shift clustering in the joint domain, with parameters $(h_s, h_r, M) = (16, 19, 40)$, where spatial regions containing less than M pixels are eliminated.

The form of the joint domain filter kernel (4.74) is reminiscent of the bilateral filter kernel (3.34–3.37) discussed in §3.2.2. The difference between mean shift and bilateral filtering, however, is that in mean shift the spatial coordinates of each pixel are adjusted along with its color values, so that the pixel migrates more quickly towards other pixels with similar colors, and can therefore later be used for clustering and segmentation.

Determining the best bandwidth parameters to use with mean shift remains somewhat of an art, although a number of approaches have been explored. These include optimizing the bias-variance

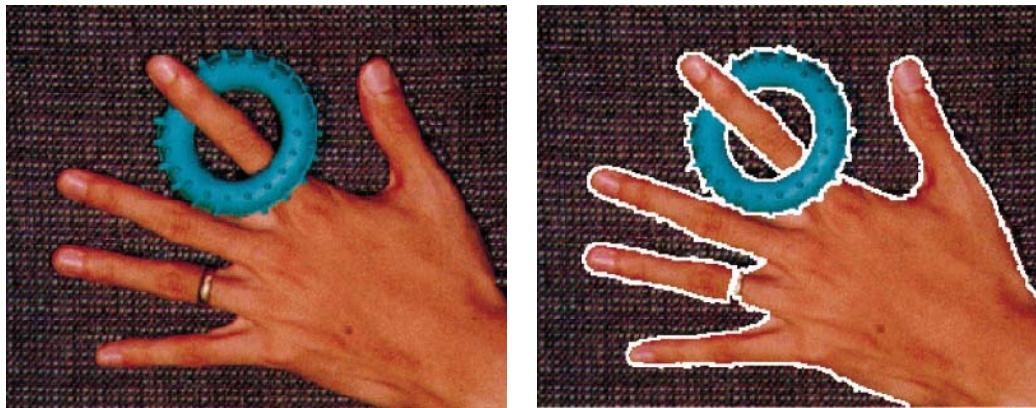


Figure 4.71: Another mean shift color image segmentation with parameters $(h_s, h_r, M) = (16, 19, 40)$ (Comaniciu and Meer 2002).

tradeoff, looking for parameter ranges where the number of clusters varies slowly, optimizing some external clustering criterion, or using top-down (application domain) knowledge (Comaniciu and Meer 2002, Comaniciu and Meer 2003). It is also possible to change the orientation of the kernel in joint parameter space for applications such as spatio-temporal (video) segmentations (Wang *et al.* 2004).

Mean shift has been applied to a number of different problems in computer vision, including face tracking, 2D shape extraction, and texture segmentation, as cited in (Comaniciu and Meer 2002), and more recently in stereo matching §10 (Wei and Quan 2004), non-photorealistic rendering §9.6 (DeCarlo and Santella 2002), and video editing §13.2.1 (Wang *et al.* 2005). Paris and Durand (2007) provide a nice review of such applications, as well as techniques for more efficiently solving the mean shift equations and producing hierarchical segmentations. I highly recommend this paper for anyone interested in mean shift segmentation.

4.5.3 Normalized cuts

While bottom-up merging techniques aggregate regions into coherent wholes, and mean-shift techniques try to find clusters of similar pixels using mode finding, the normalized cuts technique introduced by Shi and Malik (2000) examines the *affinities* (similarities) between nearby pixels and tries to separate groups that are connected by weak affinities.

Consider the simple graph shown in Figure 4.72a. The pixels in group *A* are all strongly connected with high affinities, shown as thick red lines, as are the pixels in group *B*. The connections between these two groups, shown as thinner blue lines, are much weaker. A *normalized cut* between the two groups, shown as a dashed line, separates them into two clusters.

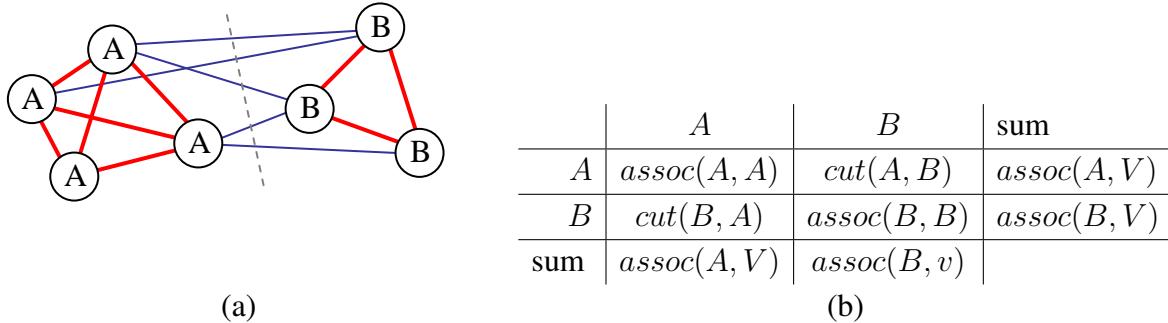


Figure 4.72: *Sample weighted graph and its normalized cut:* (a) a small sample graph and its smallest normalized cut; (b) tabular form of the associations and cuts for this graph. The assoc and cut entries are computed as area sums of the associated weight matrix \mathbf{W} (Figure 4.73). Normalizing the table entries by the row or column sums produces normalized associations and cuts $N\text{assoc}$ and $N\text{cut}$.

The cut between two groups A and B is defined as the sum of all the weights being cut,

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij}, \quad (4.75)$$

where the weights between two pixels (or regions) i and j measure their similarity. Using a minimum cut as a segmentation criterion, however, does not result in reasonable clusters, since the smallest cuts usually involve isolating a single pixel.

A better measure of segmentation is the normalized cut, which is defined as

$$N\text{cut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)}, \quad (4.76)$$

where $\text{assoc}(A, V) = \text{assoc}(A, A) + \text{cut}(A, B)$ is the sum of *all* the weights associated with nodes in A . Figure 4.72b shows how the cuts and associations can be thought of as area sums in the weight matrix $\mathbf{W} = [w_{ij}]$, where the entries of the matrix have been permuted to that the nodes in A come first and the nodes in B come second. Figure 4.73 shows an actual weight matrix for which these area sums can be computed. Dividing each of these areas by the corresponding row sum (right hand column of Figure 4.72b) results in the normalized cut and association values. These normalized values better reflect the fitness of a particular segmentation, since they look for collections of edges that are weak relative to all of the edges both inside and emanating from a particular region.

Unfortunately, computing the optimal normalized cut is NP-complete. Instead, Shi and Malik (2000) suggest computing a real-valued assignment of nodes to groups. Let \mathbf{x} be the *indicator vector* where $x_i = +1$ iff $i \in A$ and $x_i = -1$ iff $i \in B$. Let $\mathbf{d} = \mathbf{W}\mathbf{1}$ be the row sums of the symmetric matrix \mathbf{W} and $\mathbf{D} = \text{diag}(\mathbf{d})$ be the corresponding diagonal matrix. Shi and Malik

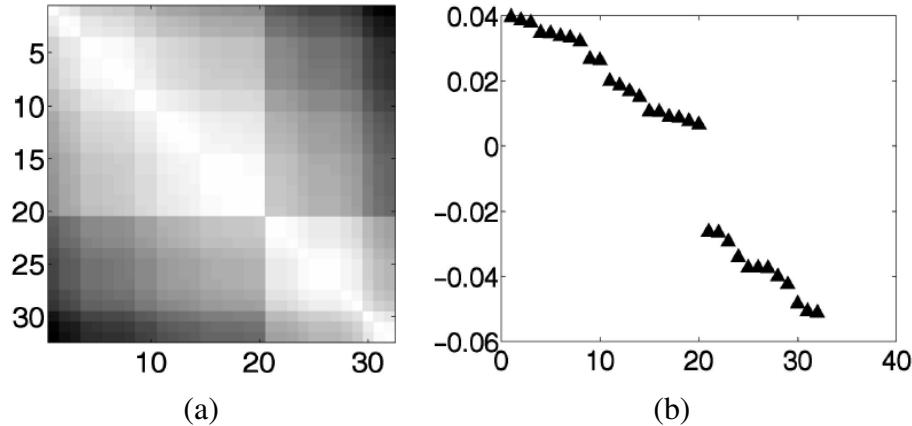


Figure 4.73: *Sample weight table and its 2nd smallest eigenvector (Shi and Malik 2000): (a) sample 32×32 weight matrix \mathbf{W} ; (b) eigenvector corresponding to the second smallest eigenvalue of the generalized eigenvalue problem $(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}$.*

(2000) show that minimizing the normalized cut over all possible indicator vectors \mathbf{x} is equivalent to minimizing

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}, \quad (4.77)$$

where $\mathbf{y} = ((\mathbf{1} + \mathbf{x}) - b(\mathbf{1} - \mathbf{x}))/2$ is a vector consisting of all 1s and $-bs$ such that $\mathbf{y} \cdot \mathbf{d} = 0$. Minimizing this *Raleigh quotient* is equivalent to solving the generalized eigenvalue system

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}, \quad (4.78)$$

which can be turned into a regular eigenvalue problem

$$(\mathbf{I} - \mathbf{N})\mathbf{z} = \lambda\mathbf{z}, \quad (4.79)$$

where $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is the *normalized* affinity matrix (Weiss 1999) and $\mathbf{z} = \mathbf{D}^{1/2}\mathbf{y}$. Because these eigenvectors can be interpreted as the large modes of vibration in a spring-mass system, normalized cuts is an example of a *spectral method* for image segmentation.

Weiss (1999) shows how normalizing the affinity matrix and then using the top k eigenvectors to reconstitute a \mathbf{Q} matrix, as proposed by Scott and Longuet-Higgins (1990) produces better segmentations than the algorithm originally proposed by Shi and Malik (2000). Other papers that extend the basic normalized cuts framework by either modifying the affinity matrix or finding better discrete solutions to the minimization problem include (Meilă and Shi 2006, Meilă and Shi 2001, Ng *et al.* 2001, Yu and Shi 2003, Tolliver and Miller 2006).

Figure 4.73b shows the second smallest (real-valued) eigenvector corresponding to the weight matrix shown in Figure 4.73a. (Here, the rows have been permuted to separate the two groups of

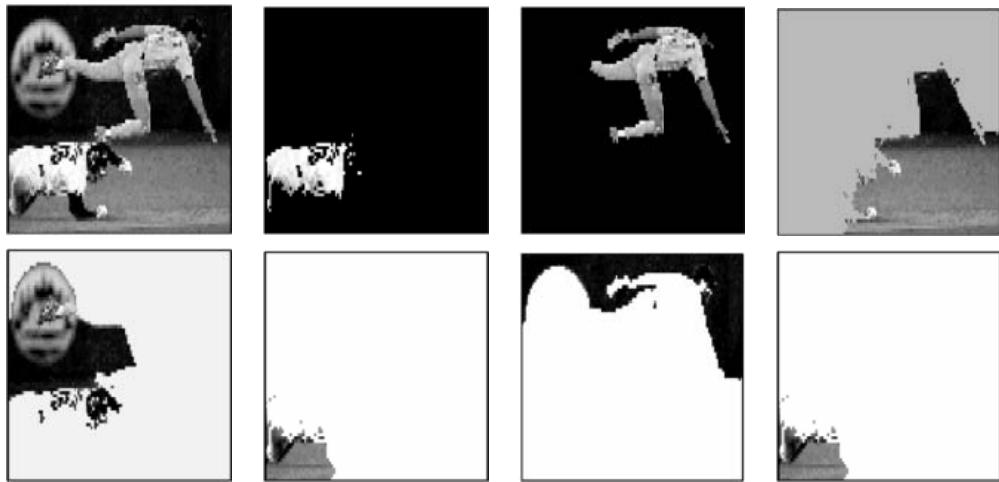


Figure 4.74: *Normalized cuts segmentation (Shi and Malik 2000), showing the input image along with the components returned by the normalized cuts algorithm.*

variables that belong to the different components of this eigenvector.) After this real-valued vector is computed, the variables corresponding to positive and negative eigenvector values are associated with the two cut components. This process can be further repeated to hierarchically subdivide an image, as shown in Figure 4.74.

The original algorithm proposed by Shi and Malik (2000) used spatial position and image feature differences to compute the pixel-wise affinities,

$$w_{ij} = \exp\left(-\frac{\|\mathbf{F}_i - \mathbf{F}_j\|^2}{\sigma_F^2} - \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_s^2}\right) \quad (4.80)$$

(for pixels within a radius $\|\mathbf{x}_i - \mathbf{x}_j\| < r$), where \mathbf{F} is a feature vector that consists of intensities, colors, or oriented filter histograms. (Note how (4.80) is the negative exponential of the joint feature space distance (4.74).)

In subsequent work Malik *et al.* (2001) look for *intervening contours* between pixels i and j and define an intervening contour weight

$$w_{ij}^{IC} = 1 - \max_{\mathbf{x} \in l_{ij}} p_{con}(\mathbf{x}), \quad (4.81)$$

where l_{ij} is the image line joining pixels i and j and $p_{con}(\mathbf{x})$ is the probability of an intervening contour perpendicular to this line, which is defined as the negative exponential of the oriented energy in the perpendicular direction. They multiply these weights with a texton-based texture similarity metric, and use an initial over-segmentation based purely on local pixel-wise features to re-estimate intervening contours and texture statistics in a region-based manner. Figure 4.75 shows the results of running this improved algorithm on a number of test images.

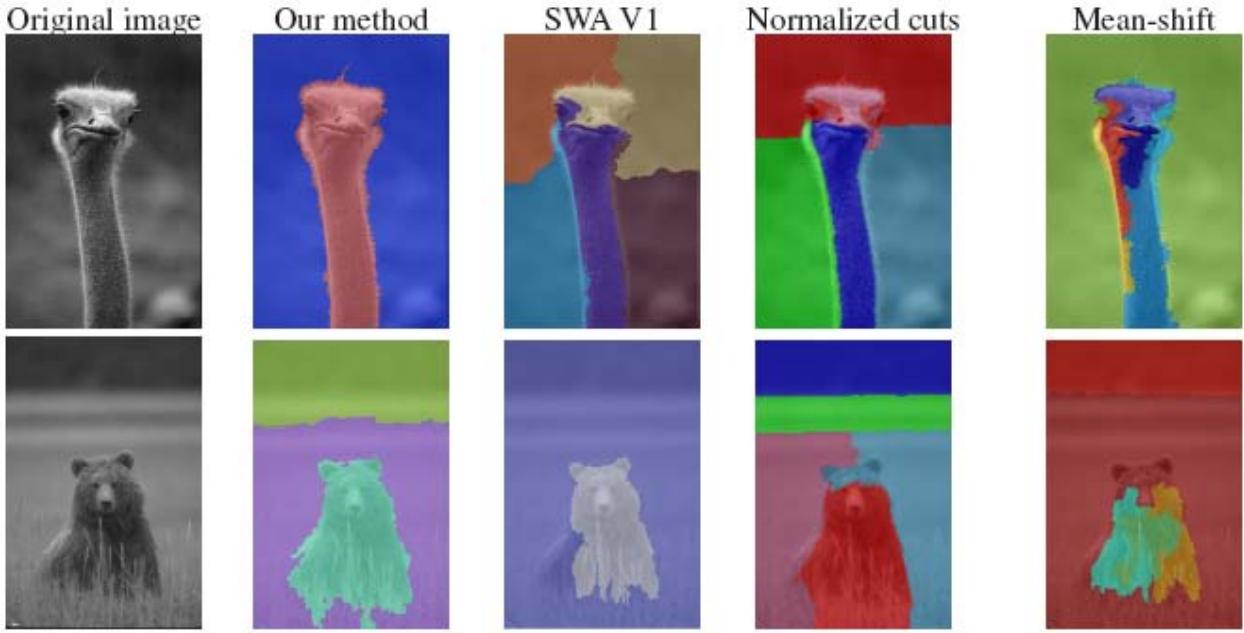


Figure 4.75: Comparative segmentation results from (Alpert et al. 2007), where “Our method” refers to the authors’ probabilistic bottom-up merging algorithm.

Because it requires the solution of large sparse eigenvalue problems, normalized cuts can be quite slow. Sharon *et al.* (2006) present a way to accelerate the computation of the normalized cuts using an approach inspired by algebraic multigrid (Brandt 1986, Briggs *et al.* 2000). To coarsen the original problem, they select a smaller number of variables such that the remaining fine-level variable are *strongly coupled* to at least one coarse-level variable. Figure 4.67 shows this process schematically, while (4.57) gives the definition for strong coupling, except that in this case, the original weights w_{ij} in the normalized cut are used instead of merge probabilities p_{ij} .

Once a set of coarse variables has been selected, an inter-level interpolation matrix with elements similar to the left hand side of (4.57) is used to define a reduced version of the normalized cuts problem. In addition to computing the weight matrix using interpolation-based coarsening, additional region statistics are used to modulate the weights in order to produce better segmentation results. Once a normalized cut has been obtained at the coarsest level of analysis, the membership values of finer-level nodes are computed by interpolating parent values and mapping values within $\epsilon = 0.1$ of 0 and 1 to pure booleans.

The resulting segmentation by weighted aggregation (SWA) algorithm produces better segmentation results than the original normalized cuts approach (Figure 4.75), although not quite as good as the authors’ most recent probabilistic bottom-up merging algorithm (Alpert *et al.* 2007), which is described in §4.5.1.

In even more recent work, Wang and Oliensis (2008) show how to estimate statistics over

segmentations (e.g., mean region size) directly from the affinity graph. They use this to produce segmentations that are more *central* w.r.t. other possible segmentations, and show that these match human segmentations better.

4.5.4 Graph cuts and energy-based methods

A common theme in image segmentation algorithms is the desire to group pixels together that have similar appearance (statistics) and to have the boundaries between pixels in different regions be of short length and across visible discontinuities. If we restrict the boundary measurements to be between immediate neighbors and compute regions membership statistics by summing over pixels, we can formulate this as a classic pixel-based energy function using either a *variational formulation* (regularization) §3.6.1 or as a binary Markov Random Field §3.6.2.

Examples of the continuous approach include (Mumford and Shah 1989, Chan and Vese 1992, Zhu and Yuille 1996, Tabb and Ahuja 1997) along with the level set approaches discussed in §4.4.3. An early example of a discrete labelling problem that combines both region-based and boundary-based energy terms is the work of Leclerc (1989), who used minimum description length (MDL) coding to derive the energy function being minimized. Boykov and Funka-Lea (2006) present a wonderful survey of various energy-based techniques for binary object segmentation, some of which I discuss below.

As we saw in the section on binary Markov Random Fields §3.6.2, the energy corresponding to a segmentation problem can be written as (c.f. (3.99) and (3.107–3.112))

$$E(f) = \sum_{i,j} E_r(i,j) + E_b(i,j), \quad (4.82)$$

where the region term

$$E_r(i,j) = E_S(I(i,j); R(f(i,j))) \quad (4.83)$$

is the negative log likelihood that pixel intensity (or color) $I(i,j)$ is consistent with the statistics of region $R(f(i,j))$, and the boundary term

$$E_b(i,j) = s_x(i,j)\delta(f(i,j) - f(i+1,j)) + s_y(i,j)\delta(f(i,j) - f(i,j+1)) \quad (4.84)$$

measures the inconsistency between \mathcal{N}_4 neighbors modulated by local horizontal and vertical smoothness terms $s_x(i,j)$ and $s_y(i,j)$.

Region statistics can be something as simple as the mean gray level or color (Leclerc 1989), in which case

$$E_S(I; \mu_k) = \|I - \mu_k\|^2. \quad (4.85)$$

Alternatively, they can be more complex, such as region intensity histograms (Boykov and Jolly 2001) or color Gaussian mixture models (Rother *et al.* 2004). For smoothness (boundary) terms, it

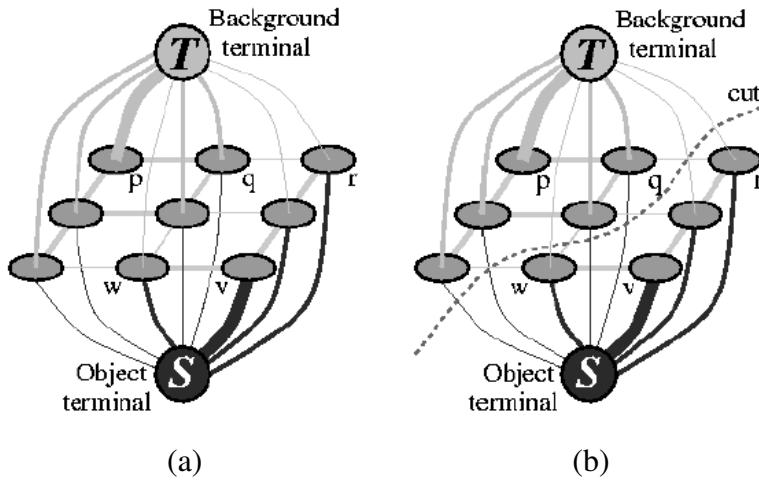


Figure 4.76: *Graph cuts for region segmentation* (Boykov and Jolly 2001): (a) energy function encoded as a max flow problem; (b) the minimum cut determines the region boundary.

[Note: Can I get a better quality original from Yuri?]

is common to make the strength of the smoothness $s_x(i, j)$ inversely proportional to the local edge strength (Boykov et al. 2001).

Originally, energy-based segmentation problems were optimized using iterative gradient descent techniques, which were slow and prone to getting trapped in local minima. Boykov and Jolly (2001) were the first to apply the binary MRF optimization algorithm developed by Greig et al. (1989) to binary object segmentation.

In their approach, the user first delineates pixels in the background and foreground regions using a few strokes of an image brush (Figure 3.64). These pixels then become the *seeds* that tie nodes in the *s-t graph* to the source and sink labels s and t (Figure 4.76a). Seed pixels can also be used to estimate foreground and background region statistics (intensity or color histograms).

The capacities of the other edges in the graph are derived from the region and boundary energy terms, i.e., pixels with higher compatibilities to the foreground or background region get stronger connections to the respective source/sink, and adjacent pixels with greater smoothness get stronger links as well. Once the min-cut/max-flow problem has been solved using a polynomial time algorithm (Goldberg and Tarjan 1988, Boykov and Kolmogorov 2001) pixels on either side of the computed cut are labelled according to the source or sink they remain connected to (Figure 4.76b). While graph cuts is just one of several known techniques for MRF energy minimization §B.6.4, it is still the one most commonly used for solving binary MRF problems.

The basic binary segmentation algorithm of Boykov and Jolly (2001) has been extended in a number of directions. The *GrabCut* system of Rother et al. (2004) iteratively re-estimates the region statistics, which are modeled as a mixtures of Gaussians in color space. This allows their

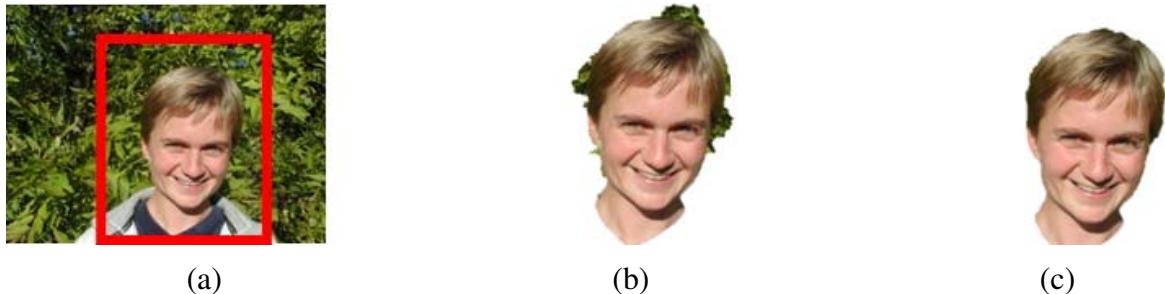


Figure 4.77: *Grab Cut* image segmentation (Rother et al. 2004): (a) the user draws a bounding box in red; (b) the algorithm guesses color distributions for object and background and performs a binary segmentation; (c) the process is repeated.

system to operate given minimal user input such as a single bounding box (Figure 4.77), in which case the background color model is initialized from a strip of pixels around the box outline. (The foreground color model is initialized from the interior pixels, but quickly converges to a better estimate of the object.) The user can also place additional strokes to refine the segmentation as the solution progresses. In more recent work, Cui *et al.* (2008) use color and edge models derived from previous segmentations of similar objects to improve the local models used in GrabCut.

Another major extension to the original binary segmentation formulation is the addition of *directed edges*, which allows boundary regions to be oriented, e.g., to prefer light to dark transitions or vice versa (Kolmogorov and Boykov 2005). Figure 4.78 shows an example where the directed graph cut correctly segments the light gray liver from its dark gray surround. The same approach can be used to measure the *flux* exiting a region, i.e., the signed gradient projected normal to the region boundary. Combining oriented graphs with larger neighborhoods enables approximating continuous problems such as those traditionally solved using levels sets in the globally optimal graph cuts framework (Boykov and Kolmogorov 2003, Kolmogorov and Boykov 2005).

Even more recent developments in graph-cut based segmentation techniques include the addition of connectivity priors to force the foreground to be in a single piece (Vicente *et al.* 2008) and shape priors to use knowledge about an object's shape during the segmentation process (Lempitsky and Boykov 2007, Lempitsky *et al.* 2008a).

While optimizing the binary MRF energy (4.82) requires the use of combinatorial optimization techniques such as max-flow, an approximate solution can be obtained by converting the binary energy terms into quadratic energy terms defined over a continuous $[0, 1]$ random field, which then becomes a classical membrane-based regularization problem (3.97–3.100). The resulting quadratic energy function can then be solved using standard linear system solvers (3.101–3.102), although if speed is an issue, you should use multigrid or one of its variants (Appendix §A.5). Once the continuous solution has been computed, it can be thresholded at 0.5 to yield a binary segmentation.

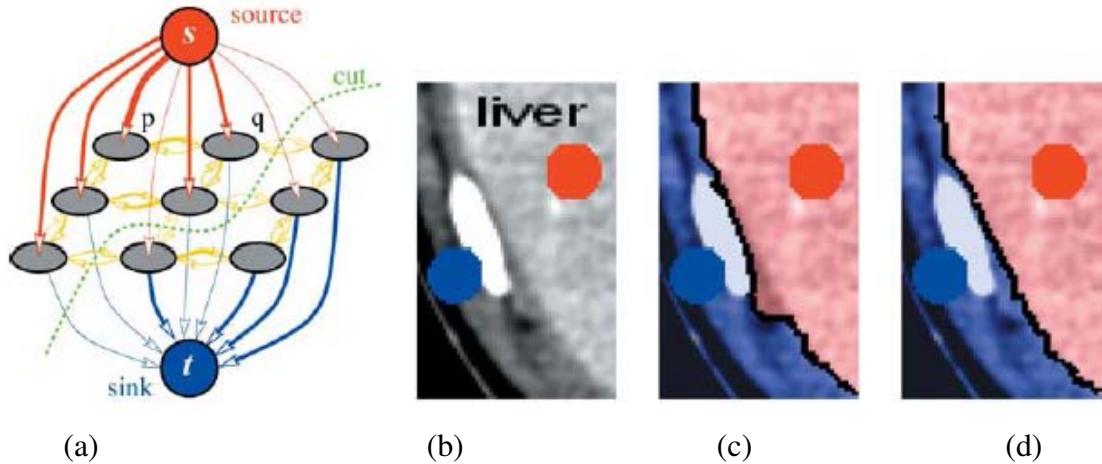


Figure 4.78: *Segmentation with a directed graph cut (Boykov and Funka-Lea 2006)*: (a) directed graph; (b) image with seed points; (c) the undirected graph incorrectly continues the boundary along the bright object; (d) the directed graph correctly segments the light gray region from its darker surround.

[Note: Can I get a better quality original from Yuri?]

The $[0, 1]$ continuous optimization problem can also be interpreted as computing the probability at each pixel that a *random walker* starting at that pixel ends up at one of the labelled seed pixels, which is also equivalent to computing the potential in a resistive grid where the resistors are equal to the edge weights (Grady 2006). K -way segmentations can also be computed by iterating through the seed labels, using a binary problem with one label set to 1 and all the others set to 0 to compute the relative membership probabilities for each pixel. In follow-on work, Grady and Ali (2008) use a precomputation of the eigenvectors of the linear system to make the solution with a novel set of seeds faster, which is related to the Laplacian matting problem presented in §9.4.3 (Levin *et al.* 2008). Singaraju *et al.* (2008) add directed edge constraints in order to support flux, which makes the energy piecewise quadratic and hence not solvable as a single linear system. The Random Walker algorithm can also be used to solve the Mumford-Shah segmentation problem (Grady and Alvino 2008) and to compute fast multigrid solutions (Grady 2008).

An even faster way to compute a continuous $[0, 1]$ approximate segmentation is to compute *weighted geodesic distances* between the 0 and 1 seed regions (Bai and Sapiro 2007), which can also be used to estimate soft alpha mattes §9.4.3.

4.5.5 Application: Medical image segmentation

One of the most promising applications of image segmentation is in the medical imaging domain, where it can be used to segment anatomical tissues for later quantitative analysis. Figure 4.78

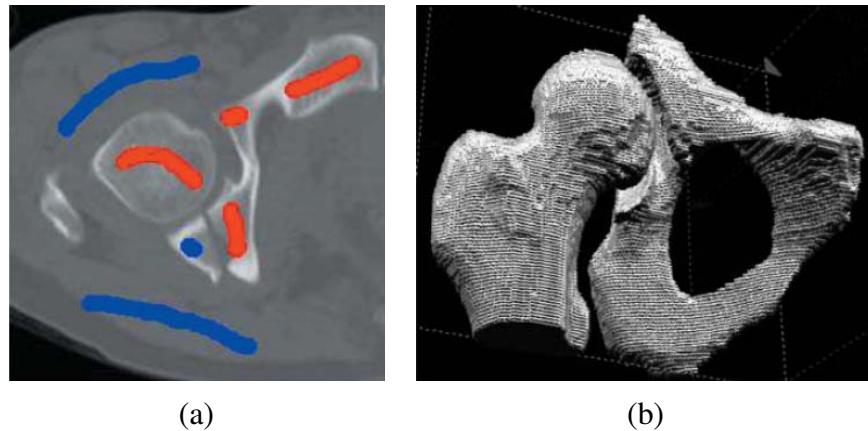


Figure 4.79: *3D volumetric medical image segmentation using graph cuts (Boykov and Funka-Lea 2006): (a) CT (computed tomography) slice with some seeds; (b) recovered 3D volumetric bone model (on a $256 \times 256 \times 119$ voxel grid).*

shows a binary graph cut with directed edges being used to segment the liver tissue (light gray) from its surrounding bone (white) and muscle (dark gray) tissue. Figure 4.79 shows the segmentation of bones in a $256 \times 256 \times 119$ CT (computed X-ray tomography) volume. Without the powerful optimization techniques available in today's image segmentation algorithms, such processing used to require much more laborious tracing of individual X-ray slices.

4.6 Exercises

Ex 4.1 (Interest point detector) Implement one or more keypoint detectors and compare their performance (with your own, or with a classmate's)

Possible detectors:

1. Laplacian or Difference of Gaussian
2. Förstner-Harris Hessian (try different formula variants given in (4.9–4.11)).
3. oriented/steerable filter, either looking for 2nd order high 2nd response, or two edges in a window (Koethe 2003), discussed in §4.1.1.
4. others from (Mikolajczyk *et al.* 2005, Tuytelaars and Mikolajczyk 2007)

Additional optional steps could include:

1. Compute the detections on a sub-octave pyramid and find 3D maxima.

2. Find local orientation estimates (using steerable filter responses or a gradient histogramming method).
3. Implement non-maximal suppression, such as the adaptive technique of [Brown et al. \(2005\)](#).
4. Vary the window shape and size (pre-filter and aggregation).

To test for repeatability, download the code from [*Note: Need Web URL here, see ([Tuytelaars and Mikolajczyk 2007](#))*] ([Mikolajczyk et al. 2005](#), [Tuytelaars and Mikolajczyk 2007](#)), or simply rotate/shear your own test images (pick a domain you may want to use later, e.g., for outdoor stitching).

Measure the stability of your scale and orientation estimates as well.

Ex 4.2 (Interest point descriptor) Implement one or more descriptors (steered to local scale and orientation) and compare their performance (with your own, or with a classmate's)

Possible descriptors:

1. contrast-normalized patches ([Brown et al. 2005](#))
2. SIFT
3. HLOG
4. others from ([Mikolajczyk and Schmid 2005](#))

Optionally do this on a sub-octave pyramid, find 3D maxima.

Vary the choose window shape and size (pre-filter and aggregation);

Ex 4.3 (Feature matcher) [*Note: The following ideas are pretty old; update them:]*

use SSD or correlation score; estimate bias/gain; use local feature vector (Schmid, Jets [[Von Der Malsburg ?](#)])

How about David Lowe's adaptive threshold idea, used in MOPS ([Brown et al. 2005](#))?

See also §7.1.2 and Ex 7.1 for more techniques and a more quantitative comparison.

Ex 4.4 (ROC curve computation) Given a pair of curves (histograms) plotting the number of matching and non-matching features as a function of Euclidean distance d as shown in Figure 4.23b, derive an algorithm for plotting an ROC curve (Figure 4.23a). In particular, let $t(d)$ be the distribution of true matches and $f(d)$ be the distribution of (false) non-matches. Write down the equations for the ROC, i.e., TPR(FPR), and the AUC.

Hint: Plot the cumulative distributions $T(d) = \int t(d)$ and $F(d) = \int f(d)$ and see if these help you derive the TPR and FPR at a given threshold θ .

```

struct SEdgel {
    float e[2][2];           // edgel endpoints (zero crossing)
    float x, y;              // sub-pixel edge position (midpoint)
    float n_x, n_y;          // orientation, as normal vector
    float theta;              // orientation, as angle (degrees)
    float length;             // length of edgel
    float strength;           // strength of edgel (local gradient magnitude)
};

struct SLine : public SEdgel {
    float line_length;        // length of line (estimated from ellipsoid)
    float sigma;               // estimated std. dev. of edgel noise
    float r;                  // line equation: x * n_y - y * n_x = r
};

```

Table 4.2: A potential C++ structure for edgel and line elements

[Note: Use a smaller font for the code. Should these kinds of code snippets be moved into Appendix C.2?]

Ex 4.5 (Image matcher—part 1) Select some set of image features (corners, lines, regions) that you can somewhat reliably match across images taken from disparate points of view. ([Lowe 1999](#), [Schaffalitzky and Zisserman 2002](#), [Sivic and Zisserman 2003](#))

Later exercise will show how to combine this with camera motion / 3D structure to completely match the images.

Ex 4.6 (Feature tracker) find corresponding points, string together; (optional: winnow out matches with epipolar geometry or 3D – need material in next 2 chapters) add new points at each frame; evaluate quality of matches, terminate if necessary.

Note that affine registration used in ([Shi and Tomasi 1994](#)) is not introduced until §[7.2](#) and Ex [7.2](#).

Ex 4.7 (Facial feature tracker) initialize tracker on person's facial features, use it to drive a simple 2D morph of another face or cartoon (reuse morphing code from exercise in previous chapter)

Ex 4.8 (Edge detector) Implement an edge detector of your choice. Compare its performance to that of other classmates', or from code downloaded from the Net.

A simple but well-performing sub-pixel edge detector can be created as follows:

1. Blur the input image a little,

$$B_\sigma(\mathbf{x}) = G_\sigma(\mathbf{x}) * I(\mathbf{x}).$$

2. Construct a Gaussian pyramid (Exercise refex:pyramid),

$$P = \text{Pyramid}\{B_\sigma(\mathbf{x})\}$$

3. Subtract an interpolated coarser-level pyramid image from the original resolution blurred image,

$$S(\mathbf{x}) = B_\sigma(\mathbf{x}) - P.\text{InterpolatedLevel}(L).$$

4. For each quad of pixels, $\{(i, j), (i + 1, j), (i, j + 1), (i + 1, j + 1)\}$, count the number of zero crossings along the four edges.
5. When there are exactly two zero crossing, compute their locations using (4.25) and store these edgel endpoints along with the midpoint in the edgel structure (Table 4.2).
6. For each edgel, compute the local gradient by taking the horizontal and vertical differences between the values of S along the zero crossing edges (Figure 4.34c). [Note: Perhaps make this a separate figure closer to the end of this chapter.]
7. Store the magnitude of this gradient as the edge strength, and either its orientation, or that of the segment joining the edgel endpoints, as the edge orientation.
8. Add the edgel to a list of edgels, or alternatively, store it in a 2D array of edgels (addressed by pixel coordinates).

Table 4.2 shows a possible representation for each computed edgel.

Ex 4.9 (Edge linking and thresholding) Link up the edges computed in the previous exercise into chains, and optionally perform thresholding with hysteresis.

Some suggested steps include:

1. Store the edgels either in a 2D array (say an integer image with indices into the edgel list), or pre-sort the edgel list first by (integer) x coordinates and then y coordinates, for faster neighbor finding.
2. Pick up an edgel from the the list of unlinked edgels, and find its neighbors in both directions until no neighbor is found, or a closed contour is obtained. Flag edgels as linked as you visit them, and push them onto your list of linked edgels.
3. Alternatively, generalize a previously developed connected component algorithm (Exercise 3.14) to perform the linking in just two raster passes.

4. [Optional] Perform hysteresis-based thresholding ([Canny 1986](#)). Starting at one end of the contour, walk along the contour until... [*Note: Re-read Canny to see if a good description exists there...*]
5. [Optional] Link together contours that have small gaps but whose endpoints have similar orientations.
6. [Optional] Find junctions between adjacent contours, e.g., using some of the ideas (or references) from ([Maire et al. 2008](#)).

Ex 4.10 (Contour matching) Convert a closed contour (linked edgel list) into its arc-length parameterization, and use this to match object outlines. [*Note: See some of Kimia's papers for examples and references.*]

Some suggested steps include:

1. Walk along the contour and create a list of (x_i, y_i, s_i) triplets, using the arc-length formula
$$s_{i+1} = s_i + \|\mathbf{x}_{i+1} - \mathbf{x}_i\|. \quad (4.86)$$
2. Resample this list onto a regular set of (x_j, y_j, j) samples using linear interpolation of each segment.
3. Compute the average value of x and y , e.g., $\bar{x} = \sum_{j=0 \dots S} x_j$. (Careful: the value of $S = \max s_i$ is generally non-integral, so adjust your formula accordingly.)
4. [Variant] Directly resample the original (x_i, y_i, s_i) piecewise linear function onto a length-independent set of samples, say $j \in [0, 1023]$. (Using a power of 2 length will make subsequent Fourier transforms more convenient.)
5. Compute the Fourier transform of the curve, treating each (x, y) pair as a complex number.
6. To compare two curves, fit a linear equation to the phase difference between the two curves. (Careful: phase wraps around at 360° . Also, you may wish to weight samples by their Fourier spectrum magnitude. See §[7.1.2](#).)
7. [Optional] Prove that the constant phase component corresponds to the temporal shift in s , while the linear component corresponds to rotation.

Of course, feel free to try any other curve descriptor and matching technique from the computer vision literature.

Ex 4.11 (Jigsaw puzzle solver–project) Write a program to automatically solve a jigsaw puzzle from a set of scanned puzzle pieces. The project may include the following components:

1. Scan the pieces (either face up or face down) on a flatbed scanner with a distinctively colored background.
2. Optionally scan in the box top to use as a low-resolution reference image.
3. Use color-based thresholding to isolate the pieces.
4. Extract the contour of each piece using edge finding and linking.
5. Optionally re-represent each contour using an arc-length or some other re-parameterization. Break up the contours into meaningful matchable pieces (hard?).
6. Optionally associate color values with each contour to help in the matching.
7. Optionally match pieces to the reference image using some rotationally invariant feature descriptors.
8. Solve a global optimization or (backtracking) search problem to snap pieces together and/or place them in the correct location relative to the reference image.
9. Test your algorithm on a succession of more difficult puzzles, and compare your results with others'.

Ex 4.12 (Artistic rendering) application: ink-and-watercolor rendering (see above)...? [Note: move this to Computational Photography chapters since we have an NPR section §9.6 there.]

Ex 4.13 (Successive approximation line detector) Successive approximation from linked edges: [Note: ... details TBD ...]

Ex 4.14 (Hough transform line detector) Implement a Hough transform for finding lines in images:

1. Create an accumulator array of the appropriate user-specified size and clear it. The user can specify the spacing in degrees between orientation bins and in pixels between distance bins. The array can either be allocated as integer (for simple counts), floating point (for weighted counts), or as an array of vectors for keeping back pointers to the constituent edges.

2. For each detected edgel at location (x, y) and orientation $\theta = \tan^{-1} n_y/n_x$, compute the value of

$$d = xn_x + yn_y \quad (4.87)$$

and increment the accumulator corresponding to (θ, d) . Optionally weight the vote of each edge by its length (see Exercise 4.8) and/or the strength of its gradient.

3. Optionally smooth the scalar accumulator array by adding in values from its immediate neighbors (this helps counteract the *discretization* effect of voting for only a single bin—see Exercise 3.7).
4. Find the largest peaks (local maxima) in the accumulator corresponding to lines.
5. For each peak, optionally re-fit the lines to the constituent edgels, using *total least squares* §A.2. In this step, optionally use the original edgel lengths and/or strength weights to weight the least squares fit, as well as potentially the agreement between the hypothesized line orientation and the edgel orientation. Determine whether these heuristics help increase the accuracy of the fit.
6. After fitting each peak, zero-out or eliminate that peak and its adjacent bins in the array, and move on to the next largest peak.

Test out your Hough transform on a variety of images taken indoors and outdoors, as well as checkerboard calibration patterns.

For the latter case (checkerboard patterns), you can modify your Hough transform by collapsing *antipodal* bins $(\theta \pm 180^\circ, -d)$ with (θ, d) to find lines that do not care about polarity changes. Can you think of examples in real world images where this might be desirable as well?

Ex 4.15 (Line fitting uncertainty) Estimate the uncertainty (covariance) in your line fit using uncertainty analysis.

1. After determining which edgels belong to the line segment (using either successive approximation or Hough transform), re-fit the line segment using total least square (Van Huffel and Vandewalle 1991), i.e., find the mean/centroid of the edgels and then use eigenvalue analysis to find the dominant orientation.
2. Compute the perpendicular errors (deviations) to the line and robustly estimate the variance of the fitting noise using an estimator such as MAD §B.3.
3. Optionally re-fit the line parameters by throwing away outliers or using a robust norm / influence function.

4. Estimate the error in the perpendicular location of the line segment and its orientation

[Note: Need to work this out in more detail, and perhaps include the error analysis in the main text.]

Ex 4.16 (Vanishing points) Compute vanishing points and refine line equations. The results will be used later to track a target (Ex 5.7) or reconstruct architecture (Ex ???)

Ex 4.17 (Vanishing point uncertainty) Perform an uncertainty analysis on your estimated vanishing points. You will need to decide how to represent your vanishing point, e.g., homogeneous coordinates on a sphere, to handle VPs near infinity.

See the discussion of Bingham distributions in (Collins and Weiss 1990).

Ex 4.18 (Snake evolution) Prove that in the absence of external forces, a snake will always shrink to a small circle and eventually a single point, regardless of whether first or second order smoothness (or some combination) is used.

Hint: If you can show that the evolution of the $x(s)$ and $y(s)$ components are independent, you can analyze the 1-D case more easily.

Ex 4.19 (Snake tracker) use B-splines (or not); use efficient LDU solver (introduced in §6 and Appendix A.4);

Ex 4.20 (Intelligent scissors) write an intelligent scissor cutout function:

Ex 4.21 (Region segmentation) Implement one of the region segmentation algorithms described in §4.5. Some popular segmentation algorithms include:

- k-means §4.5.2;
- mixtures of Gaussians §4.5.2;
- mean shift §4.5.2 and Exercise 4.22;
- normalized cuts §4.5.3;
- similarity graph-based segmentation §4.5.1;
- binary Markov Random Fields solved using graph cuts §4.5.4.

Apply your region segmentation to a video sequence and use it to track moving regions from frame to frame.

Alternatively, test out your segmentation algorithm on the Berkeley segmentation database (Martin *et al.* 2001).

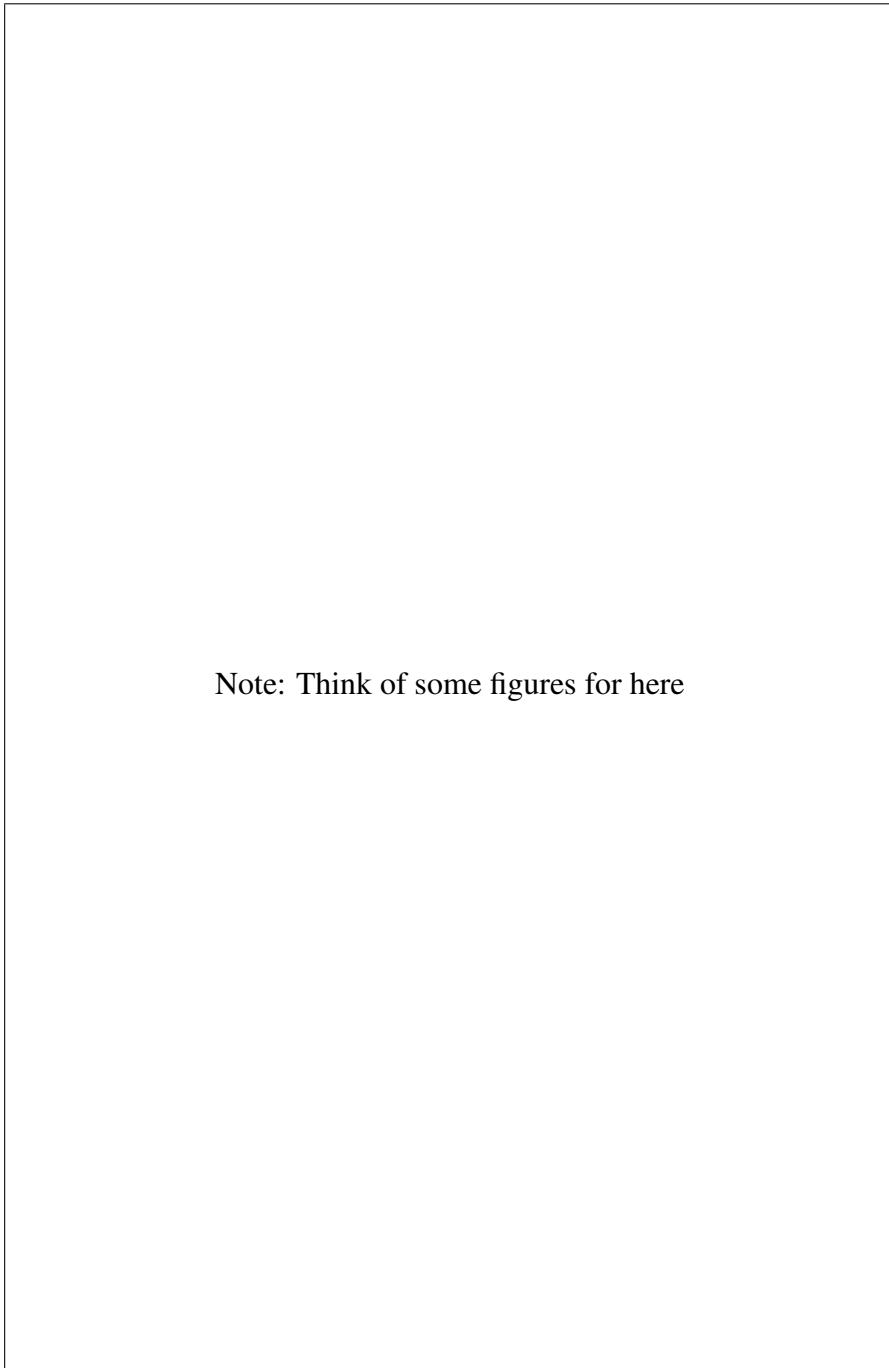
Ex 4.22 (Mean shift) Develop a mean shift segmentation algorithm for color images (Comaniciu and Meer 2002).

1. Convert your image to Lab space, or keep the original RGB colors, and augment these with the pixel (x, y) locations
2. For every pixel (L, a, b, x, y) , compute the weighted mean of its neighbors using either a unit ball (Epanechnikov kernel) or finite-radius Gaussian, or some other kernel of your choosing. Weight the color and spatial scale differently, e.g., using values of $(h_s, h_r, M) = (16, 19, 40)$ as shown in Figure 4.71.
3. Replace the current value with this weighted mean and iterate until either the motion is below a threshold or a fininte number of steps has been taken.
4. Cluster all final values (modes) that are within a threshold together, i.e., find the connected components. Since each pixel is associated with a final mean shift (mode) value, this results in an image segmentation, i.e., each pixel is labelled with its final component.
5. [Optional] Only use a random subset of the pixels as starting points, and find which component each unlabelled pixels belongs to either by finding its nearest neighbor or by iterating the mean shift until it finds a neighboring track of mean shift values. Describe the data structures you use to make this efficient.
6. [Optional] Mean shift divides the kernel density function estimate by the local weighting to obtain a step size that is guaranteed to converge but may be slow. Use an alternative step size estimation algorithm from the optimization literature to see if you can make the algorithm converge faster.

Chapter 5

Geometric alignment and calibration

5.1	Feature-based alignment	311
5.1.1	Least squares	311
5.1.2	<i>Application:</i> Panography	312
5.1.3	Iterative algorithms	312
5.1.4	<i>Application:</i> Photo stabilization	314
5.1.5	Robust least squares and RANSAC	314
5.1.6	3D alignment	316
5.1.7	Uncertainty modeling	317
5.2	Pose estimation (extrinsic calibration)	317
5.2.1	Linear algorithms	317
5.2.2	Iterative algorithms	318
5.2.3	<i>Application:</i> Videomouse	318
5.3	Geometric (intrinsic) calibration	318
5.3.1	Calibration patterns	319
5.3.2	Vanishing points and lines	320
5.3.3	<i>Application:</i> Single View Metrology	320
5.3.4	Rotational motion	320
5.3.5	Radial distortion	320
5.4	Exercises	321



Note: Think of some figures for here

Figure 5.1: *Some examples of geometric alignment and calibration: ...*
[Note: TODO]

Even though chapter is mostly about calibration, start with the *pose estimation* problem first, since it's simpler to explain. [Note: Is this really the case?]

[Note: Re-read (*Tsai 1987*) (look for your hardcopy) since it has a nice literature review]

5.1 Feature-based alignment

5.1.1 Least squares

Simplest examples to introduce: how to register two 2D images with a translation, then a scaled rotation, then affine transformation (or simpler: use motion model hierarchy)

[Note: Move Table 7.1 and its associated text from §7.2 to here.]

[Note: The following text is copied from the stitching TR:]

Once we have computed a set of matched feature point correspondences, the next step is to estimate the motion parameters \mathbf{p} that best register the two images. The usual way to do this is to use least squares, i.e., to minimize the sum of squared residuals given by (5.11),

$$E_{\text{LS}} = \sum_i \|\mathbf{r}_i\|^2 = \|\tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) - \hat{\mathbf{x}}'_i\|^2. \quad (5.1)$$

Many of the motion models presented in §2.1.2, i.e., translation, similarity, and affine, have a *linear* relationship between the motion and the unknown parameters \mathbf{p} .¹ In this case, a simple linear regression (least squares) using normal equations $\mathbf{A}\mathbf{p} = \mathbf{b}$ works well. [Note: For poorly conditioned problems, it is sometimes preferable to use QR decomposition on the set of linear equations instead of normal equation (*Golub and Van Loan 1996*). However, such conditions rarely arise in image registration.]

Uncertainty weighting and robust regression. The above least squares formulation assumes that all feature points are matched with the same accuracy. This is often not the case, since certain points may fall in more textured regions than others. If we associate a variance estimate σ_i^2 with each correspondence, we can minimize *weighted least squares* instead,

$$E_{\text{WLS}} = \sum_i \sigma_i^{-2} \|\mathbf{r}_i\|^2. \quad (5.2)$$

As discussed in §7.1.3, a covariance estimate for patch-based matching can be obtained by multiplying the inverse of the Hessian with the per-pixel noise estimate (7.42). Weighting each squared

¹ 2D Euclidean motion can be estimated with a linear algorithm by first estimating the cosine and sine entries independently, and then normalizing them so that their magnitude is 1.

residual by the inverse covariance $\Sigma_i^{-1} = \sigma_n^{-2} \mathbf{A}_i$ (which is called the *information matrix*), we obtain

$$E_{\text{CWLS}} = \sum_i \|\mathbf{r}_i\|_{\Sigma_i^{-1}}^2 = \sum_i \mathbf{r}_i^T \Sigma_i^{-1} \mathbf{r}_i = \sum_i \sigma_n^{-2} \mathbf{r}_i^T \mathbf{A}_i \mathbf{r}_i, \quad (5.3)$$

where \mathbf{A}_i is the *patch Hessian* (7.57).

If there are outliers among the feature-based correspondences (and there almost always are), it is better to use a robust version of least squares, even if an initial RANSAC or MLS stage has been used to select plausible inliers. The robust least squares cost metric (analogous to (7.2)) is then

$$E_{\text{RLS}}(\mathbf{u}) = \sum_i \rho(\|\mathbf{r}_i\|_{\Sigma_i^{-1}}). \quad (5.4)$$

As before, a commonly used approach to minimize this quantity is to use iteratively re-weighted least squares, as described in §7.1.3.

5.1.2 Application: Panography

Use a simple translational + scaled rotation model (optimization from §5.1.1) together with averaging to create *panographs*. Easy and fun!

Recently automated by [Nomura et al. \(2007\)](#) and [Zelnik-Manor and Perona \(2007\)](#).

Add pointer to Exercise 5.2.

Simple 2-image alignment (t only or $t+sR$) is a trivial least squares.

Multi-image is trickier: can reference everything to one base image, or can just reference to each other. To avoid matrix inverses, better to have each T reference to a global coordinate system.

Gotcha: degeneracies in the system, since 4 dof gauge freedom. Make this a problem in the exercise and ask for solutions.

5.1.3 Iterative algorithms

This is a good place to introduce non-linear regression (non-linear least squares).

Use it first to estimate a rotation (either constrain $c^2 + s^2 = 1$, or use θ as a parameter).

Use it to describe how to do homography.

[Note: Introduce compositional approach ([Szeliski and Shum 1997](#), [Baker and Matthews 2004](#))? Not needed for feature-based, since no significant precomputation of gradients.]

How much of the material should be deferred to Appendix A.3?

[Note: The following text is copied from the stitching TR:]

For *non-linear* measurement equations such as the homography given in (7.53), rewritten here as

$$\hat{x}' = \frac{(1 + h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \quad \text{and} \quad \hat{y}' = \frac{h_{10}x + (1 + h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}, \quad (5.5)$$

an iterative solution is required to obtain accurate results. An initial guess for the 8 unknowns $\{h_{00}, \dots, h_{21}\}$ can be obtained by multiplying both sides of the equations through by the denominator, which yields the linear set of equations,

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \quad (5.6)$$

However, this is not optimal from a statistical point of view, since the denominator can vary quite a bit from point to point.

One way to compensate for this is to *re-weight* each equation by the inverse of current estimate of the denominator, D ,

$$\frac{1}{D} \begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \quad (5.7)$$

While this may at first seem to be the exact same set of equations as (5.6), because least squares is being used to solve the over-determined set of equations, the weightings *do* matter and produce a different set of normal equations that performs better in practice (with noisy data). [Note: In the book, give an example of computing the homography given six point correspondences (two keystoned interlocking planes).]

The most principled way to do the estimation, however, is to directly minimize the squared residual equations (5.11) using the Gauss-Newton approximation, i.e., performing a first-order Taylor series expansion in \mathbf{p} , which yields,

$$\hat{\mathbf{x}}'_i - \tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) = \mathbf{J}_{\mathbf{x}'} \Delta \mathbf{p} \quad (5.8)$$

or

$$\begin{bmatrix} \hat{x}' - \tilde{x}' \\ \hat{y}' - \tilde{y}' \end{bmatrix} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\tilde{x}'x & -\tilde{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\tilde{y}'x & -\tilde{y}'y \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}. \quad (5.9)$$

While this looks similar to (5.7), it differs in two important respects. First, the left hand side consists of unweighted *prediction errors* rather than pixel displacements, and the solution vector is a *perturbation* to the parameter vector \mathbf{p} . Second the quantities inside $\mathbf{J}_{\mathbf{x}'}$ involve *predicted* feature locations (\tilde{x}', \tilde{y}') instead of *sensed* feature locations (\hat{x}', \hat{y}') . Both of these are subtle and yet they lead to an algorithm that, when combined with proper checking for downhill steps (as in the Levenberg-Marquardt algorithm), will converge to a minimum. (Iterating the (5.7) equations is not guaranteed to do so, since it is not minimizing a well-defined energy function.)

The above formula is analogous to the *additive* algorithm for direct registration since the change to the *full* transformation is being computed §7.2. If we prepend an incremental homography to the current homography instead, i.e., we use a *compositional* algorithm, we get $D = 1$ (since $\mathbf{p} = 0$) and the above formula simplifies to

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}, \quad (5.10)$$

where I have replaced (\tilde{x}', \tilde{y}') with (x, y) for conciseness. (Notice how this results in the same Jacobian as (7.65).)

[Note: (Zhang 1998b) and (Hartley and Zisserman 2004) also contain similar derivations. Check these and incorporate.]

5.1.4 Application: Photo stabilization

Use feature-based direct alignment to stabilize a set of photographs (redundant with the Video Stabilization application §7.2.1).

Use the translation plus rotation, which requires iterative least squares.

Useful to detect best image, use in PhotoMontage (cut and paste).

Optionally do slow in-between dissolves or morphs for subtle animations.

5.1.5 Robust least squares and RANSAC

[Note: Need better intro text, since this used to follow the feature matching section in the alignment TR.]

Once an initial set of feature correspondences has been computed, we need to find a set that is will produce a high-accuracy alignment. One possible approach is to simply compute a least squares estimate or to use a robustified (iteratively re-weighted) version of least squares, as discussed in §5.1.3. However, in many cases, it is better to first find a good starting set of *inlier* correspondences, i.e., points that are all consistent with some particular motion estimate.²

Two widely used solution to this problem are called RANdom SAmple Consensus, or RANSAC for short (Fischler and Bolles 1981) and *least median of squares* (LMS) (Rousseeuw 1984). Both techniques start by selecting (at random) a subset of k correspondences, which is then used to compute a motion estimate \mathbf{p} , as described in §5.1.3. The *residuals* of the full set of correspondences

² For pixel-based alignment methods, §7.1.1, hierarchical (coarse-to-fine) techniques are often used to lock onto the *dominant motion* in a scene.

are then computed as

$$\mathbf{r}_i = \tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) - \hat{\mathbf{x}}'_i, \quad (5.11)$$

where $\tilde{\mathbf{x}}'_i$ are the *estimated* (mapped) locations, and $\hat{\mathbf{x}}'_i$ are the sensed (detected) feature point locations.

The RANSAC technique then counts the number of *inliers* that are within ϵ of their predicted location, i.e., whose $\|\mathbf{r}_i\| \leq \epsilon$. (The ϵ value is application dependent, but often is around 1-3 pixels.) Least median of squares finds the median value of the $\|\mathbf{r}_i\|$ values.

The random selection process is repeated S times, and the sample set with largest number of inliers (or with the smallest median residual) is kept as the final solution. Either the initial parameter guess \mathbf{p} or the full set of computed inliers is then passed on to the next data fitting stage. In a more recently developed version of RANSAC called PROSAC (PROgressive SAmple Consensus), random samples are initially added from the most “confident” matches, thereby speeding up the process of finding a (statistically) likely good set of inliers (Chum and Matas 2005).

[Note: Probably omit the following:

To ensure that the random sampling has a good chance of finding a true set of inliers, a sufficient number of trials S must be tried. Let p be the probability that any given correspondence is valid, and P be the total probability of success after S trials. The likelihood in one trial that all k random samples are inliers is p^k . Therefore, the likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S \quad (5.12)$$

and the required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}. \quad (5.13)$$

Stewart (1999) gives the following examples of the required number of trials S to attain a 99% probability of success:

k	p	S
3	0.5	35
6	0.6	97
6	0.5	293

As you can see, the number of trials grows quickly with the number of sample points used. This provides a strong incentive to use the minimum number of sample points k possible for any given trial, which in practice is how RANSAC is normally used.] [Note: In the book, have a whole section (Appendix?) on robust statistics. Also, talk about other applications, such as (Torr and Murray 1997).]

Name	Formula	# D.O.F.	Common situation	Reference
translation	$\mathbf{x}' = \mathbf{x} + \mathbf{t}$	2	small local translation	§2.1.2, §7.2
2D rigid	$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}$	3	small motion and in-plane rotation	§2.1.2, §7.2
similarity	$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t}$	4	in-plane rotation and zoom/loom	§2.1.2, §7.2
affine	$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{t}$	6	small planar motions and orthography	§2.1.2, §7.2
projective	$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$	8	planar motions	§2.1.2, §8.1
3D rigid (epipolar)	$\tilde{\mathbf{x}}'^T \mathbf{F} \tilde{\mathbf{x}} = 0$	5–8	3D rigid object motion	§6.2, §10.1

Table 5.1: *Potential motion models that can be used to constrain feature point matches. The 3D rigid motion model (a.k.a. epipolar constraint) has a number of reduced d.o.f. variants, discussed in §6.2.*

Geometric constraints and RANSAC

[Note: This is older text from the book: re-integrate with the above. Table 2.1 seems worth keeping, even though it somewhat duplicates Table 2.1.]

Sometimes we know a particular *motion model* applies to the feature motion, and we can use this to constrain the matches. Such models will be introduced in Chapters 6 and 8, but we can summarize some of the most common ones below (see also Table 2.1).

The basic idea is to choose an initial set of matches (often the *minimal* set is chosen) to estimate the motion, and then re-compute the number of inliers (RANSAC) or robust matching error (Least Median Squares). Several random starting configurations can be selected for robustness.

[Note: Defer full discussion to §6.2.1?]

Once a motion model has been chosen, the set of correspondences can be re-computed using a constrained motion model (Beardsley *et al.* 1996)? (Hartley and Zisserman 2004)?

5.1.6 3D alignment

Easy cases first: translation and affine.

Introduce incremental rotation for updates (derivatives w.r.t. only 3 parameters).

See text in Stitching TR to see if want to borrow any of this.

Closed form solutions (Procrustes and the other one (Horn))

Iterated closest point algorithms (Besl and McKay 1992, Zhang 1994); also, octree-based computation of distance functions (Szeliski and Lavallee 1996).

5.1.7 Uncertainty modeling

First natural place to discuss uncertainty.

Show how covariance is approximated (Cramer-Rao lower bound) by the inverse Hessian in Levenberg-Marquardt.

Show how uncertainty depends a lot on geometric structure of world. For example, tracking points on table top is a bad way to do object insertion (match move) far off the table (e.g., show little tree, try to hang ornaments on them).

5.2 Pose estimation (extrinsic calibration)

when 3D points are known

also known as *extrinsic calibration*

usually follows intrinsic calibration (§5.3) in practice, but simpler math here, so introduce it first

5.2.1 Linear algorithms

Direct Linear Transform (DLT) goes back to Sutherland's Sketchpad

Sutherland, I. E. (1963). Sketchpad: A man-machine graphical communication system. In Proceedings AFIPS Spring Joint Computer Conference, volume 23, pages 329–346, Detroit, Michigan.

I. Sutherland. Sketchpad—a man-machine graphical communication system. Technical Report 296, Lincoln Laboratory, Massachusetts Institute of Technology, 1963. (also, his Ph.D. thesis)

[Note: Is it earlier in the Photogrammetry community? Look in the latest version of the Manual (*Slama 1980*) (2004).]

[Note: Triangulation is even easier, since do not have to do rotation. Give a forward pointer to §6.1.]

Mention that under orthography, there is not even a perspective division, so it's just regular least squares (explain this case first). Show why it's optimal, whereas perspective is not if we cross-multiply.

Try writing equations as

$$\begin{aligned}\hat{\mathbf{i}} \cdot \mathbf{R}_j(\mathbf{p}_i - \mathbf{c}_j) &= x_{ij} \\ \hat{\mathbf{j}} \cdot \mathbf{R}_j(\mathbf{p}_i - \mathbf{c}_j) &= y_{ij}\end{aligned}$$

Obvious that this is linear in \mathbf{p}_i .

algebraic algorithms: (DeMenthon and Davis 1995) (Quan and Lan 1999) (Ameller *et al.* 2000)
 (submitted to ECCV'2000 but not accepted, from Trigg's pub page <http://www.inrialpes.fr/movi/people/Triggs/home.html>
 - has it appeared in print somewhere by now?)

from orthogonal lines (see above, e.g., VideoMouse, Criminisi, ...)

(Horn 1987): closed form quaternion fitting

Check out (Lorusso *et al.* 1995); see Golub96 p. 601 for Procrustes algorithm.

5.2.2 Iterative algorithms

This is a good place to introduce non-linear regression (non-linear least squares).

Show how we can minimize re-projection error in DLT using (1) re-weighting and (2) true non-linear LS. (See UW course notes...)

Factored representation: $\mathbf{K}(\mathbf{R}|t)$.

Introduce incremental rotation for updates (derivatives w.r.t. only 3 parameters).

Tsai (Tsai 1987); Bogart's view correlation (Bogart 1991); Gleicher's TLL camera control (Gleicher and Witkin 1992)

How much of the material should be deferred to Appendix A.3?

5.2.3 Application: Videomouse

(Hinckley *et al.* 1999): note that they use dots instead of a checkerboard, which is why I've moved it here instead of after vanishing lines...

Is it too specialized, hard to reproduce? OK if we just print up a checkerboard pattern and wave it in front of a Web cam.

[Note: Should we put it after pose estimation??? No: introduce it here, since direct application of line extraction and vanishing point computation. Full implementation is deferred to exercise in the next section.]

do model fitting (Lowe, Drummond) - try Rubik cube mouse, or ball version (ellipses).

Note that the WII uses a bar (or is it a T?) of IR LEDs that are viewed by a very high frame-rate camera. With a Web-camera, implement a simpler version of the 3D mouse that does this.

[Note: Why isn't there an ambiguity in the tilt around the bar if it's just a bar? Check out the CMU grad student who's worked on this...]

5.3 Geometric (intrinsic) calibration

EXIF tags: give rough camera specs

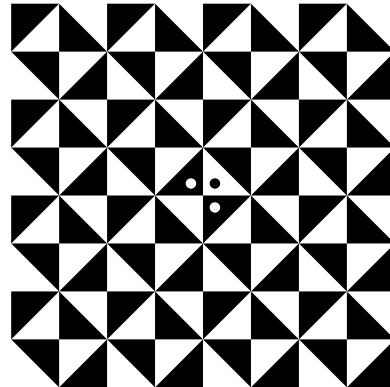


Figure 5.2: *Sample calibration pattern, which can be used for radial distortion estimation, focal length calibration, and also PSF / blur estimation.* [Note: Replace with newer one from (Joshi *et al.* 2008).]

relationship of f_{mm} (focal length in mm) to f (focal length in pixels, or unit-less (NDC)) to field of view

5.3.1 Calibration patterns

known points: Tsai's method (Tsai 1987), optimal estimation (+ uncertainty models)

forward pointer to pose estimation §5.2.2, where iterative optimization is explained in the simple case.

unknown points: forward pointer to SfM (§6) and autocalibration (self-calibration) techniques

Planar calibration patterns

N-planes splines calibration (Champeboux *et al.* 1992b) or (Champeboux *et al.* 1992a) (read & compare)

also Gremban (Gremban *et al.* 1988) and (Grossberg and Nayar 2001) (rixels) (the above techniques are equivalent to a 3D calibration grid)

Photogrammetrists use surveyed points in a field (Slama 1980, Atkinson 1996).

Debevec's technique (which paper, 2001?): use pins pushed into a board in front of the camera. Useful for locating optical center (front nodal point) and amount of deviation from pinhole model.

Zhang's multi-plane technique (Zhang 1999); this one is quite different, since do not know pose of plane for each picture. It's essentially a vanishing point technique.

Use a pattern with checks and triangles, plus some dots near middle to compute relative orientation, if desired (e.g., rotating camera).

5.3.2 Vanishing points and lines

(Caprile and Torre 1990, Becker and Bove 1995, Shum *et al.* 1998, Cipolla *et al.* 1999, Antone and Teller 2000, Criminisi *et al.* 2000, Rother 2002, Schindler *et al.* 2008)

Use vanishing points to establish the 3 coordinate axes. (If two of the vanishing points are finite (not at infinity), can guess the focal length if we assume known optic center)

(Antone and Teller 2000) cites (McLean and Kotturi 1995, Shufelt 1999, Tuytelaars *et al.* 1997) (see papers/ directory)

re-read (Antone and Teller 2000) to get latest (and more) references

5.3.3 Application: Single View Metrology

[Note: This could also go into the Architecture section §11.7.1]

Loosely based on Criminisi *et al.* (2000)

Look at Steve's course notes to see how he teaches this.

[Note: The following should go into an exercise:]

Use vanishing points to establish the 3 coordinate axes. (If two of the vanishing points are finite (not at infinity), can guess the focal length if we assume known optic center)

Click on a few ground-plane lines to establish the horizon (other ground plane lines automatically assigned. Use color coding for direction and value (elevation).

Click on points for incidence relations.

Export to 3D VRML.

Optional: texture map planes.

5.3.4 Rotational motion

focal length from rotation (Hartley 1994a, Stein 1995, Stein 1997, de Agapito *et al.* 1999, Kang and Weiss 1999, Shum and Szeliski 2000, Frahm and Koch 2003) (see image alignment section) (is (Hartley 1997) the journal version of (Hartley 1994a)?)

For more general motion, (Hartley *et al.* 2000) deals with searching for the plane at infinity and has a good literature review. That article also cites (Hartley 1997, Hartley 1998) (self-calibration and chirality).

5.3.5 Radial distortion

Plumb line method (Brown 1971, Kang 2001, El-Melegy and Farag 2003):

1. Extract (nearly) straight lines §4.3.

2. Adjust image parameters until all lines are straight.

Kang's method ([Kang 2001](#)) has user draw snakes near straight lines, then optimizes their positions and radial distortion.

If unknown scene, can measure distance of each edge point to straight line fit.

If known grid, can measure grid points to their projected locations, minimizing the distortion in the center of the image (may need to fit a homography as well for the plane tilt) ([Tsai 1987](#)).

[Note: Develop the grid extraction code either here or in the previous chapter (vanishing points). The only trick is that for radial distortion, cannot use Hough to get vanishing points and lines. Need to link into splines first.]

Bundle adjustment: simultaneous determination of camera intrinsics and *unknown* 3D structure (see s.f.m. §[6](#)) ([Stein 1997](#), [Sawhney and Kumar 1997](#), [Hsu and Sawhney 1998](#))).

Chromatic aberration: different radial distortion functions in each color band: can put colors back together. However, there are two problems with this simple model: Light isn't really just made up of RGB, but of frequencies in between, so that yellow will shift one way for red and another for green (in fact, it will disperse / mis-focus, since each wavelength will shift a different amount). Secondly, chromatic aberration also causes mis-focus ([§2.2.3](#)). Make this into an exercise, and see what the students can observe, e.g., from sharp in-focus B/W and colored calibration targets.

5.4 Exercises

Ex 5.1 (Feature-based image alignment) Take a set of photos of an action scene or portrait (preferably in motor-drive / continuous shooting mode) and align them to make a composite or flip-book animation.

1. Extract features and feature descriptors using some of the techniques described in the previous chapter §[4.1.1–4.1.2](#).
2. Match your features using nearest neighbor matching with a nearest neighbor distance ratio test ([4.18](#)) §[4.1.3](#).
3. Compute an optimal 2D translation and rotation between the first image and all subsequent images, using least squares §[5.1.1](#) with optional RANSAC for robustness §[5.1.5](#).
4. Resample all of the images onto the first image's coordinate frame §[3.5.1](#) using either bilinear or bicubic resampling.

“Stabilize” to the first image (or compute average of translations and rotation angles) and resample all the images.

Crop to a common area.

Turn into an animated GIF (using software available on Web)

Ex 5.2 (Panography) Use a simple motion model (rigid or scaled rigid), feature-based alignment, and a simple compositor (average or translucent over).

Warp the images to a larger canvas.

Average the images together.

Optionally write a simple tool to let the user adjust the ordering, opacity, and add or remove images.

Discuss the limitations of this approach.

Ex 5.3 (2D match move) With a Web cam, take a picture of a magazine or book page.

Outline a figure or picture on the page with a rectangle.

Match features in this area with each new image frame.

Replace the original image with an “advertizing” insert.

Can also do this with a sporting event clip (e.g., indoor or outdoor soccer) and do billboard replacement.

Ex 5.4 (3D joystick) Get out your old Rubik’s cube (or get one from your parents).

Write a program to detect the centers of each colored square. This may work better if you find the lines on each face, and compute common vanishing points.

Estimate the rotation angle and focal length.

Use the 3D rotation and position to control a VRML viewer (is this doable?)

Ex 5.5 (Radial distortion) Implement a plumb line algorithm; (give more details from text).

Does not require any calibration image: find curved line segments, fit line segments, minimize difference w/ parametric distortion.

Calibration image (grid): find corners and enumerate (may be easier to find regions first, or polarity-changing line segments)

Optional: fit general spline model (for severe distortion)

Ex 5.6 (Chromatic aberration) Use the radial distortion estimates for each color channel computed in the previous exercise to clean up wide angle lens images (cheap cameras) with straightening and chromatic aberration.

Ex 5.7 (3D video mouse) Use the vanishing point algorithm developed in Ex 4.16 *or* the region tracker developed in Ex 4.21 to estimate the focal length, and then to track a hand-held grid (VideoMouse) or Rubik’s cube. Use this to control the 3D orientation of an object in a video game or 3D graphics application.

Ex 5.8 (Target-based calibration–omit?) Construct a calibration pattern with known 3D locations (not that easy).

Find the corners (e.g., line finder and intersect found lines).

Implement Tsai/Bogart's bundle adjuster. (Give hints on how to use the sparse solver.)

Ex 5.9 (Rotation-based calibration) Take an outdoor sequence (or indoor sequence with very little parallax).

Make sure you have taken out the radial distortion.

Track some points from frame-to-frame.

Compute the focal length, either using a small subsequence (2 or 3 frames), or a complete 360° sequence.

Discuss which one is likely to be more accurate. Optionally plot accuracy as a function of number of images used.

Ex 5.10 (Calibration accuracy) Compare the three calibration techniques presented above: plane-based, 3D target based, and rotation-based.

Easiest way is to have a different student implement each one.

Use a synthetic data set so that you know the ground truth (use the software you developed for Ex 2.3).

Assume a medium-wide focal length (say 45°).

For plane-based technique, generate a 2D grid target, project it at different inclinations.

For 3D target, create an inner cube corner, position it so it fill most of field of view.

For rotation, scatter points uniformly on a sphere until get similar number of points as other techniques.

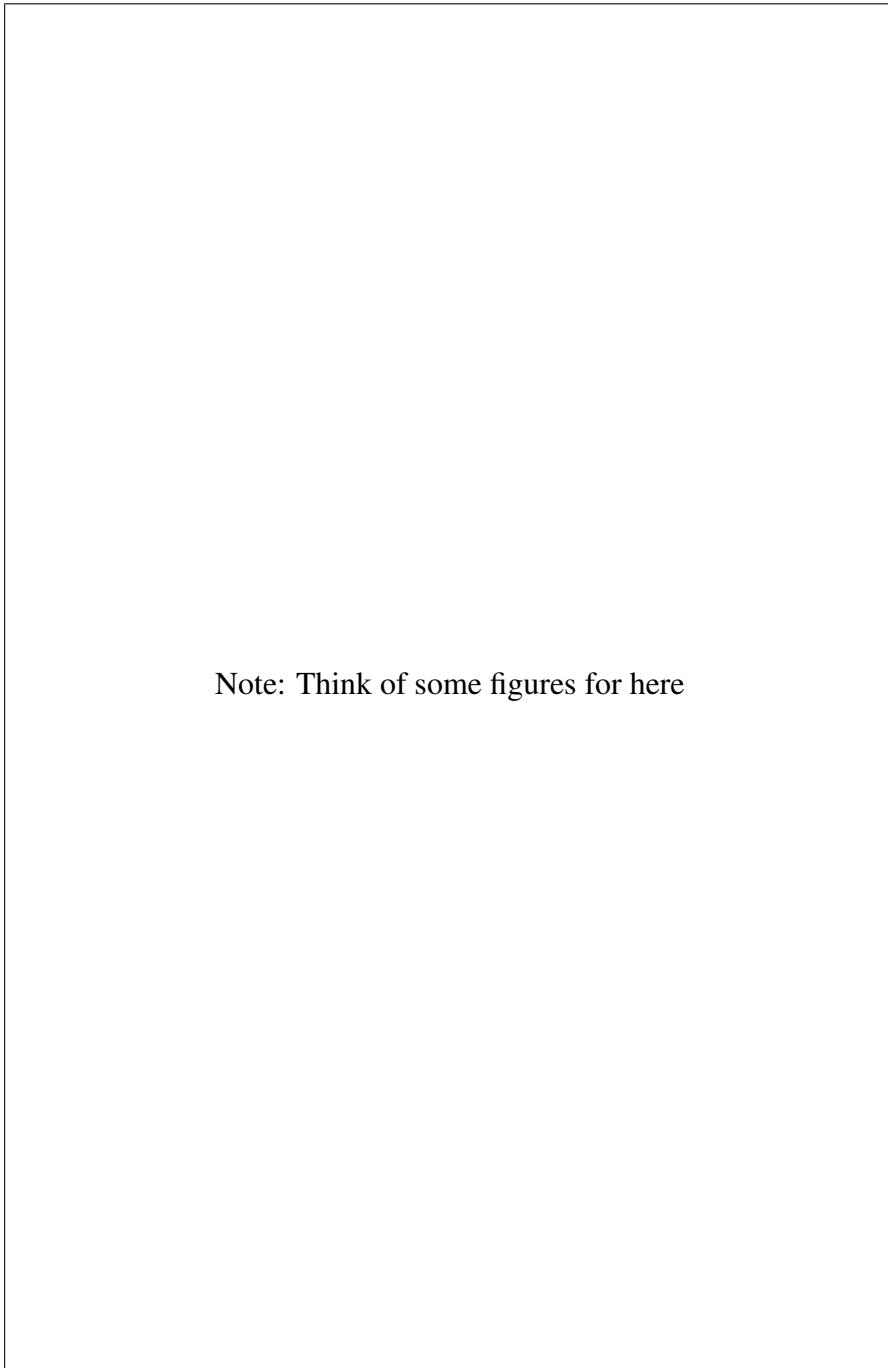
Before doing the last part, predict which of the three techniques will be the most accurate (normalize by the square root of the number of points sensed).

Add uniform noise (1 pixel). Recover camera parameters. Compare to ground truth. Re-run enough times (random noise) to get RMS error measures.

Chapter 6

Structure from motion

6.1	Triangulation	327
6.2	Two-frame structure from motion	327
6.2.1	RANSAC (revisited)	328
6.2.2	Self-calibration	328
6.2.3	<i>Application:</i> View morphing	329
6.3	Factorization	329
6.3.1	Projective factorization	330
6.3.2	<i>Application:</i> Sparse 3D model extraction	330
6.4	Bundle adjustment	330
6.4.1	Uncertainty and ambiguities	331
6.4.2	<i>Application:</i> Match move	331
6.5	Constrained structure and motion	331
6.5.1	Line and plane-based techniques	331
6.6	Exercises	332



Note: Think of some figures for here

Figure 6.1: *Some examples of structure from motion: ...*

[Note: TODO]

In previous chapter, 3D structure was assumed to be known. In this chapter, we study how to simultaneously estimate the camera pose and 3D structure of a scene, given only sparse correspondences between image features.

[Note: Some general notes and ideas:]

very rich and well studied subject

(Hartley and Zisserman 2004) is mostly devoted to this topic, as is (Faugeras and Luong 2001)

See these books for more in-depth information, as well as Oliensis' "Critique of Structure from Motion" (Oliensis 2000).

Oliensis' algorithms: read (Oliensis and Genc 2001) and references therein.

General problem formulation:

The world consists of N 3D points seen from M cameras (or equivalently, in M frames of a moving camera video), $\{\mathbf{x}_{ji}, j = 1 \dots M, i = 1 \dots N\}$ (only a subset of the points may be visible in any given frame). [Note: \mathbf{x}_{ij} would be more standard, but we will have already introduced $i = 1 \dots N$ in the pose estimation part. Also, I want to reserve j and k for the two frame indices.]

We wish to recover the location of the 3D points, $\{\mathbf{p}_i, i = 1 \dots N\}$, and the pose (and optionally the intrinsics) of the cameras, $\{\mathbf{R}_j, \mathbf{t}_j, \mathbf{K}_j, j = 1 \dots M\}$. (Note: sometimes, only the \mathbf{P}_j are recovered.)

6.1 Triangulation

Address this simpler problem first, even though it's relatively easy to solve.

Linear technique: simple closed form, but biased by distance to points.

Non-linear technique: easy problem, but trickier than expected: solution may lie behind one or both cameras.

Analytic solution (Hartley and Sturm 1997): solve cubic equation. Unfortunately, has the same problems with *chirality* (Hartley 1998). Also, does this technique work for more than two points?

Best solution may just be to do constrained optimization.

6.2 Two-frame structure from motion

(Ullman 1979): first article using "structure from motion", name stuck (but what did it do?)

E (Longuet-Higgins 1981) and F (Faugeras 1992, Faugeras 1993); what's in this one: (Luong and Faugeras 1997)? robust sampling (Huber 1981, Torr and Murray 1997, Zhang 1998a, Zhang 1998b)

[Note: (*Girod et al. 2000*, pp. 40–41) has a very nice simple formulation and diagram. Use this as the basis. May need to define a new term for the calibrated pixel coordinate $\mathbf{K}_j^{-1}\mathbf{x}_{ji}$.]

Estimation steps:

1. re-scale or otherwise adjust pixel coordinates
2. estimate \mathbf{E} or \mathbf{F} using (weighted?) least squares
3. apply SVD to find rank 2 matrix (unit eigenvalues for \mathbf{E})
4. recover epipole (translation) and rotation (or projection matrix)

Variants:

1. RANSAC for finding best matches (*Torr and Murray 1997*)
2. cubic constraint on rank, matrix reparameterization (*Torr and Murray 1997*)
3. re-weighted least squares (*Zhang 1998a, Zhang 1998b*) [Note: read both and select which one]
4. normal equation solution

Torr and Fitzgibbon's latest critique (CVPR'2003), Invariant Fitting of Two View Geometry or “In Defiance of the 8 Point Algorithm”: solve a smaller eigenvalue problem by fixing the norm of the upper 2×2 part of \mathbf{F} (works better, except for affine geometries).

David Nistér's 5-point algorithm, (see citation in (*Nistér 2003*) to CVPR 2003 paper), which he claims is more stable than other variants under difficult conditions (but requires 10th order root solver).

6.2.1 RANSAC (revisited)

Use random sampling consensus (RANSAC) (*Fischler and Bolles 1981*) to select a good set of *inliers* (*Torr and Murray 1997*) or least-median squares ((*Zhang 1998a*)?) (Concepts introduced in §5.1.5.)

6.2.2 Self-calibration

Reread latest techniques, including Cipolla's f adjustment to make \mathbf{E} the right form. Which techniques are two-frame?

6.2.3 Application: View morphing

Show how to do an automated view morph using features and triangulation.

Or, instead of triangulation, can do a sparse interpolator (RBF).

Maybe better in IBR chapter?

6.3 Factorization

(Tomasi and Kanade 1992, Costeira and Kanade 1995, Poelman and Kanade 1997, Morita and Kanade 1997, Morris and Kanade 1998, Anandan and Irani 2002) (+ perspective correction (Christy and Horaud 1996) and full projective (Triggs 1996))

Orthographic or weak perspective projection: let \mathbf{M}_j be the upper 2×4 portion of the projection matrix \mathbf{P}_j , i.e.,

$$\mathbf{P}_j = \begin{bmatrix} \mathbf{M}_j \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Then

$$\mathbf{x}_{ji} = \mathbf{M}_j \mathbf{p}_i \quad (6.1)$$

(note that should be using $\bar{\mathbf{p}}_i$, the augmented 4-vector). In matrix form

$$\hat{\mathbf{X}} = \begin{bmatrix} \mathbf{x}_{11} & \cdots & \mathbf{x}_{1i} & \cdots & \mathbf{x}_{1N} \\ \vdots & & \vdots & & \vdots \\ \mathbf{x}_{j1} & \cdots & \mathbf{x}_{ji} & \cdots & \mathbf{x}_{jN} \\ \vdots & & \vdots & & \vdots \\ \mathbf{x}_{M1} & \cdots & \mathbf{x}_{Mi} & \cdots & \mathbf{x}_{MN} \end{bmatrix} = \begin{bmatrix} \mathbf{M}_1 \\ \vdots \\ \mathbf{M}_j \\ \vdots \\ \mathbf{M}_M \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 & \cdots & \mathbf{p}_i & \cdots & \mathbf{p}_N \end{bmatrix} = \hat{\mathbf{M}} \hat{\mathbf{S}}. \quad (6.2)$$

$\hat{\mathbf{X}}$ is called the *measurement* matrix, and $\hat{\mathbf{M}}$ and $\hat{\mathbf{S}}$ the *motion* and *structure* matrices, respectively (Tomasi and Kanade 1992). [Note: Check (Tomasi and Kanade 1992) for terminology and notation. Make a note to the reader about the different notation we are using, to be consistent with the rest of the book. Note that we could also use $\hat{\mathbf{P}}_j$ instead of \mathbf{M}_j , and drop the hats from $\hat{\mathbf{X}}, \hat{\mathbf{M}}, \hat{\mathbf{S}}$.]

Estimation steps:

1. form $\hat{\mathbf{X}}$ matrix, apply SVD
2. find the Euclidean or scaled orthography solution (may also be para-perspective (Poelman and Kanade 1997))
3. optional (before forming $\hat{\mathbf{X}}$ matrix: re-scale by certainties in the measurements (Anandan and Irani 2002))

6.3.1 Projective factorization

perspective correction (Christy and Horaud 1996)

full projective (Triggs 1996)

missing data (Huynh *et al.* 2003)

convergence issues: (Oliensis and Hartley 2007)

[Note: Where does Carsten Rother's work on linear methods once rotation is known fit in?
 ((Rother 2003) is newer work, points back to older work.)]

6.3.2 Application: Sparse 3D model extraction

Show how to build a point cloud and triangulated point cloud from a factorization solution. Provide a hand-held scan of object (long focal length) as sample sequence, and include same data for exercise.

6.4 Bundle adjustment

(Triggs *et al.* 1999) bundle adjustment review—re-read carefully, a real goldmine

optimal motion estimation (Weng *et al.* 1993);

efficient versions: skyline sparse (Szeliski and Kang 1994), Schur complement (Triggs *et al.* 1999, Hartley and Zisserman 2004)

Sparse Bundle Adjustment package of (Lourakis and Argyros 2004) (cite includes URL).

on-line version (Azarbayejani and Pentland 1995, McLauchlan 2000, Davison 2003)

partitioned version (Steedly *et al.* 2003)

large (uncalibrated) collections (Fitzgibbon *et al.* 1998, Koch *et al.* 2000, Pollefeys and Van Gool 2002, Schaffalitzky and Zisserman 2002, Tuytelaars and Van Gool 2004, Snavely *et al.* 2006).

[Note: Add an application below (after Match move) on Matching internet photo collections, *a la Photo Tourism?*]

[Note: Add an application on robot navigation / SLAM, mentioning Georg Klein's most recent work on Fast-SLAM?]

[Note: Mention global approaches using the L_∞ norm [Kahl and Hartley, PAMI 2008? - not yet issued]]

Efficiency issues in structure from motion (Steedly and Essa 2001, Steedly *et al.* 2003, Snavely *et al.* 2008a)

6.4.1 Uncertainty and ambiguities

(Szeliski and Kang 1997) ; newer work on gauge ambiguities (Triggs *et al.* 1999); did I also review a paper by Daniel Morris on this topic? Oliensis has more recent and comprehensive work; also, relationship to field of view (see Teller's overview paper)

McLaughlan has done extensive work on gauge, as has Kanatani

Bartoli's latest ICCV paper: (Bartoli 2003)

Is this the right place to address chirality (Hartley 1998), or earlier? Enforcing chirality too early may lead to an un-necessarily stiff system.

6.4.2 Application: Match move

Track a long sequence (walk through campus) and add some synthetic objects. As with other applications, provide the data and make this an exercise.

See (Roble 1999), any articles by Fitzgibbon?

6.5 Constrained structure and motion

6.5.1 Line and plane-based techniques

Line-based techniques

what does (Schmid and Zisserman 1997) do?

trifocal tensors (Fitzgibbon and Zisserman 1998);

bundle-adjustment with large directional uncertainty (Szeliski and Kang 1994) ((Seitz and Anandan 1999) probably not relevant, but check)

constrained orientations (Shum *et al.* 1998, Criminisi *et al.* 2000)

line-based (Bartoli and Sturm 2003); Favaro show how to avoid T-junctions (Favaro *et al.* 2003)

note that line features can improve the performance of view morphing applications (Z. Zhang, Photo Redraw: not published? ... talk to Zhengyou, possibly Zhang-Shan-SMILE00 (sparse stereo...))

Plane-based techniques

algebraic solutions; plane + parallax; hallucination (Szeliski and Torr 1998) (resubmit to journal?)

Geometric (structural) constraints

parallelism and perpendicularity (Shum *et al.* 1998, Criminisi *et al.* 2000, Criminisi *et al.* 2000); co-planarity (homographies) (Szeliski and Torr 1998)

[Note: Do we want to do an application to architectural modeling here? Probably not, since we will later have model-based reconstruction §11.7.1. Put a forward pointer to that work here.]

Specialized motion models

[Note: Move this / merge with image stitching chapter...]

Pure rotation (homographies): useful technique for camera calibration §5.3.4; fundamental concept in mosaics (Chapter 8).

Many people have looked at rotational motion, including: (Hartley 1994a) (is this the same as (Hartley 1997)?); feature-based (check Capel for more): from §8.2.5 (Zoghliami *et al.* 1997, Capel and Zisserman 1998, Badra *et al.* 1998, McLauchlan and Jaenicke 2002) [Note: Check the date on McLaughlan ref: 2001 or 2002?] (some of these techniques use bundle adjustment, but so does our original work with Harry (Shum and Szeliski 2000))

[Note: Who has done mosaics with parallax using Bundle Adjustment? McLauchlan? RissXXX (could not read my handwriting!)]

axial rotation: shape-from-rotation (Szeliski 1991b, Fitzgibbon *et al.* 1998),
concentric mosaics (ask Harry if any work done on hand-held CMs)

6.6 Exercises

Ex 6.1 (Triangulation) Extend/modify pose estimator to now make the structure the unknown. Use DLT / linear technique to initialize. Generate synthetic data to test. See if any of the failure modes reported in (Hartley and Sturm 1997, Hartley 1998) occur in practice.

Ex 6.2 (Essential and fundamental matrix) Implement the two-frame F and E matrix estimation techniques, with suitable re-scaling for better noise.

Optional: implement improved F and/or E matrix algorithm, e.g., using Zhengyou's better metrics, Torr's 7-parameter F, RANSAC, LMS, Torr's latest re-normalization, Nister's 5-point algorithm.

Ex 6.3 (View morphing and interpolation) Application: implement the two-view automatic view morphing (between adjacent views of an object, e.g., an “object movie”).

Build a 3D triangulated model from this solution (use planar Delaunay triangulation of central view?)

Extract texture maps from most frontal view (see §11.8 for more details).

Need a triangle renderer, e.g., OpenGL or D3D. Make this an optional exercise in image processing section.

Smoothly interpolate the original views, or have interactive user control.

Ex 6.4 (Factorization) Implement factorization algorithm, using point tracks from previous chapter. (Provide some test sequences, some with tracks given as results.)

Optional: implement uncertainty rescaling? But when is this relevant?

Ex 6.5 (Perspective factorization) Implement one of the perspective improvements.

Ex 6.6 (Bundle adjuster) Implement a full bundle adjuster.

Take the Tsai/Bogart bundle adjuster from previous chapter, and add the 3D points as unknowns now.

Try alternating between solutions vs. solving jointly (using the sparse solver).

Provide a long image sequence (rotational table) for testing.

Ex 6.7 (Match move) Use the results of the previous exercise to superimpose a rendered 3D model on top of video.

Check for how “locked down” the objects are.

Ex 6.8 (Line-based reconstruction) Augment the previously developed bundle adjuster to include lines, possibly with known 3D orientations.

Optionally use co-planar pairs of lines to hypothesize planes, check which parts are consistent (Schaffalitzky and Zisserman 2002, Robertson and Cipolla 2002)

Ex 6.9 (Flexible bundle adjuster) Design a bundle adjuster that allows for arbitrary chains of transformations and prior knowledge about the unknowns. (Journal paper to be written next summer?)

Ex 6.10 (Unordered image matching) Solve the “bag of photos” problem to determine pose.

Generate a system that automatically view interpolates between key poses.

Provide subsets of IBR sequences (Bellevue Botanical, Dan Ling home) for testing? Or, use Internet data sets, like in Photo Tourism (Snavely *et al.* 2006).

Ex 6.11 (Structure without motion) Devise some algorithm(s) to handle very long sequences ((Steedly *et al.* 2003) and earlier work).

Discuss this in main text?

[Note: Try to identify some open research problems for graduate students to work on...]

Chapter 7

Dense motion estimation

7.1	Translational alignment	338
7.1.1	Hierarchical motion estimation	341
7.1.2	Fourier-based alignment	342
7.1.3	Incremental refinement	346
7.2	Parametric motion	351
7.2.1	<i>Application:</i> Video stabilization	356
7.3	Spline-based motion	356
7.3.1	<i>Application:</i> Automated morphing	359
7.4	Optical flow	359
7.4.1	Spatio-temporal filtering	360
7.4.2	<i>Application:</i> Frame interpolation	360
7.5	Layered motion	360
7.6	Learned motion models (tracking, revisited)	362
7.6.1	<i>Application:</i> Motion-based user interface	362
7.7	Exercises	362



Figure 7.1: Some examples of motion estimation: (a–b) regularization-based optical flow (Nagel and Enkelmann 1986); (c–d) layered motion estimation (Wang and Adelson 1994); (e–f) sample image and ground truth flow from evaluation database (Baker et al. 2007).

[Note: Replace one of these with learned motion (Black et al. 1997)?]

Algorithms for aligning images and estimating motion in video sequences are among the most widely used in computer vision. For example, frame-rate image alignment is used in every camcorder or digital camera that has an “image stabilization” feature.

An early example of a widely-used image registration algorithm is the patch-based translational alignment (optical flow) technique developed by [Lucas and Kanade \(1981\)](#). Variants of this algorithm are used in almost all motion-compensated video compression schemes such as MPEG and H.263 ([Le Gall 1991](#)). Similar parametric motion estimation algorithms have found a wide variety of applications, including video summarization ([Bergen et al. 1992](#), [Teodosio and Bender 1993](#), [Kumar et al. 1995](#), [Irani and Anandan 1998](#)), video stabilization ([Hansen et al. 1994](#)), and video compression ([Irani et al. 1995](#), [Lee et al. 1997](#)). More sophisticated image registration algorithms have also been developed for medical imaging and remote sensing—see ([Brown 1992](#), [Zitov'aa and Flusser 2003](#), [Goshtasby 2005](#), [Szeliski 2006a](#)) for some surveys of image registration techniques.

[Note: Skim the ([Brown 1992](#), [Goshtasby 2005](#), [Benosman and Kang 2001](#)) surveys and books to see what I've missed; also, have a look at Chuck Stewart's *Image Registration* course, <http://www.cs.rpi.edu/~stewart/>.]

[Note: Give some historical background, starting with ([Netravali and Robbins 1979](#)) (did not read), ([Huang 1981](#)), ([Horn and Schunck 1981](#)), ([Lucas and Kanade 1981](#)), ...? Probably not, unless have more reading section @ end...]

To estimate the motion between two or more images, a suitable *error metric* must first be chosen to compare the images ([§7.1](#)). Once this has been established, a suitable *search* technique must be devised. The simplest technique is to exhaustively try all possible alignments, i.e., to do a *full search*. In practice, this may be too slow, so *hierarchical* coarse-to-fine techniques ([§7.1.1](#)) based on image pyramids can be used. Alternatively, Fourier transforms ([§7.1.2](#)) can be used to speed up the computation.

To get sub-pixel precision in the alignment, *incremental* methods ([§7.1.3](#)) based on a Taylor series expansion of the image function are often used. These can also be applied to *parametric motion models* ([§7.2](#)), which model global image transformations such as rotation or shearing. For more complex motions, piecewise parametric *spline motion models* ([§7.3](#)) can be used. If pixel-accurate correspondences are desired, general-purpose *optical flow* (aka *optic flow*) techniques have been developed ([§7.4](#)). For more complex motions that include a lot of occlusions, *layered motion models* ([§7.5](#)), which decompose the scene into coherently moving layers, can work well. Finally, motion estimation can also be made more reliable by *learning* the typical dynamics or motion statistics of the scenes or objects being tracked, e.g., the natural gait of walking people ([§7.6](#)).

In this chapter, I describe each of these techniques in more detail. Additional details can be found in review and comparative evaluation papers on motion estimation ([Barron et al. 1994](#),

Mitiche and Boutheny 1996, Stiller and Konrad 1999, Szeliski 2006a, Baker *et al.* 2007).

7.1 Translational alignment

The simplest way to establish an alignment between two images is to shift one image relative to the other. Given a *template* image $I_0(\mathbf{x})$ sampled at discrete pixel locations $\{\mathbf{x}_i = (x_i, y_i)\}$, we wish to find where it is located in image $I_1(\mathbf{x})$. A least-squares solution to this problem is to find the minimum of the *sum of squared differences* (SSD) function

$$E_{\text{SSD}}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 = \sum_i e_i^2, \quad (7.1)$$

where $\mathbf{u} = (u, v)$ is the *displacement* and $e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)$ is called the *residual error* (or the *displaced frame difference* in the video coding literature).¹ (I ignore for the moment the possibility that parts of I_0 may lie outside the boundaries of I_1 or be otherwise not visible.)

In general, the displacement \mathbf{u} can be fractional, so a suitable interpolation function must be applied to image $I_1(\mathbf{x})$. In practice, a bilinear interpolant is often used, but bicubic interpolation should yield slightly better results. Color images can be processed by summing differences across all three color channels, although it is also possible to first transform the images into a different color space or to only use the luminance (which is often done in video encoders).

Robust error metrics. We can make the above error metric more robust to outliers by replacing the squared error terms with a robust function $\rho(e_i)$ (Huber 1981, Hampel *et al.* 1986, Black and Anandan 1996, Stewart 1999) to obtain

$$E_{\text{SRD}}(\mathbf{u}) = \sum_i \rho(I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)) = \sum_i \rho(e_i). \quad (7.2)$$

The robust norm $\rho(e)$ is a function that grows less quickly than the quadratic penalty associated with least squares. One such function, sometimes used in motion estimation for video coding because of its speed, is the *sum of absolute differences* (SAD) metric, i.e.,

$$E_{\text{SAD}}(\mathbf{u}) = \sum_i |I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)| = \sum_i |e_i|. \quad (7.3)$$

However, since this function is not differentiable at the origin, it is not well suited to gradient-descent approaches such as the ones presented in §7.1.3.

¹ The usual justification for using least squares is that it is the optimal estimate with respect to Gaussian noise. See the discussion below on robust alternatives as well as Appendix B.3.

Instead, a smoothly varying function that is quadratic for small values but grows more slowly away from the origin is often used. [Black and Rangarajan \(1996\)](#) discuss a variety of such functions, including the *Geman-McClure* function,

$$\rho_{\text{GM}}(x) = \frac{x^2}{1 + x^2/a^2}, \quad (7.4)$$

where a is a constant that can be thought of as an *outlier threshold*. An appropriate value for the threshold can itself be derived using robust statistics ([Huber 1981](#), [Hampel *et al.* 1986](#), [Rousseeuw and Leroy 1987](#)), e.g., by computing the *median of absolute differences*, $MAD = \text{med}_i|e_i|$, and multiplying by 1.4 to obtain a robust estimate of the standard deviation of the non-outlier noise process ([Stewart 1999](#)). [Note: Look at the *MUSE algorithm*, [Miller & Stewart, CVPR'96 \(\[54\] in \(Stewart 1999\)\)](#) to see if it should be cited here or in the section on Robust estimation.]

Spatially varying weights. The error metrics above ignore the fact that for a given alignment, some of the pixels being compared may lie outside the original image boundaries. Furthermore, we may want to partially or completely downweight the contributions of certain pixels. For example, we may want to selectively “erase” some parts of an image from consideration, e.g., when stitching a mosaic where unwanted foreground objects have been cut out. For applications such as background stabilization, we may want to downweight the middle part of the image, which often contains independently moving objects being tracked by the camera.

All of these tasks can be accomplished by associating a spatially varying per-pixel weight value with each of the two images being matched. The error metric then becomes the weighted (or *windowed*) SSD function,

$$E_{\text{WSSD}}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u})[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2, \quad (7.5)$$

where the weighting functions w_0 and w_1 are zero outside the valid ranges of the images.

If a large range of potential motions is allowed, the above metric can have a bias towards smaller overlap solutions. To counteract this bias, the windowed SSD score can be divided by the overlap area

$$A = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u}) \quad (7.6)$$

to compute a *per-pixel* (or mean) squared pixel error. The square root of this quantity is the *root mean squared* intensity error

$$RMS = \sqrt{E_{\text{WSSD}}/A} \quad (7.7)$$

often seen reported in comparative studies.

Bias and gain (exposure differences). Often, the two images being aligned were not taken with the same exposure. A simple model of linear (affine) intensity variation between the two images is the *bias and gain* model,

$$I_1(\mathbf{x} + \mathbf{u}) = (1 + \alpha)I_0(\mathbf{x}) + \beta, \quad (7.8)$$

where β is the *bias* and α is the *gain* (Lucas and Kanade 1981, Gennert 1988, Fuh and Maragos 1991, Baker *et al.* 2003b). The least squares formulation then becomes

$$E_{\text{BG}}(\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - (1 + \alpha)I_0(\mathbf{x}_i) - \beta]^2 = \sum_i [\alpha I_0(\mathbf{x}_i) + \beta - e_i]^2. \quad (7.9)$$

Rather than taking a simple squared difference between corresponding patches, it becomes necessary to perform a *linear regression*, which is somewhat more costly. Note that for color images, it may be necessary to estimate a different bias and gain for each color channel to compensate for the automatic *color correction* performed by some digital cameras.

A more general (spatially-varying non-parametric) model of intensity variation, which is computed as part of the registration process, is presented in (Jia and Tang 2003). This can be useful for dealing with local variations such as the *vignetting* caused by wide-angle lenses. It is also possible to pre-process the images before comparing their values, e.g., by using band-pass filtered images (Burt and Adelson 1983b, Bergen *et al.* 1992) or using other local transformations such as histograms or rank transforms (Cox *et al.* 1995, Zabih and Woodfill 1994), or to maximize *mutual information* (Viola and Wells III 1997, Kim *et al.* 2003). [Note: Check out (Negahdaripour and Yu 1993), A generalized brightness change model for computing optical flow.]

[Note: Add an image from (Viola and Wells III 1997) or even make an application out of medical image registration?]

Correlation. An alternative to taking intensity differences is to perform *correlation*, i.e., to maximize the *product* (or *cross-correlation*) of the two aligned images,

$$E_{\text{CC}}(\mathbf{u}) = \sum_i I_0(\mathbf{x}_i)I_1(\mathbf{x}_i + \mathbf{u}). \quad (7.10)$$

At first glance, this may appear to make bias and gain modeling unnecessary, since the images will prefer to line up regardless of their relative scales and offsets. However, this is actually not true. If a very bright patch exists in $I_1(\mathbf{x})$, the maximum product may actually lie in that area.

For this reason, *normalized cross-correlation* is more commonly used,

$$E_{\text{NCC}}(\mathbf{u}) = \frac{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0] [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]}{\sqrt{\sum_i [I_0(\mathbf{x}_i) - \bar{I}_0]^2 [I_1(\mathbf{x}_i + \mathbf{u}) - \bar{I}_1]^2}}, \quad (7.11)$$

where

$$\overline{I_0} = \frac{1}{N} \sum_i I_0(\mathbf{x}_i) \quad \text{and} \quad (7.12)$$

$$\overline{I_1} = \frac{1}{N} \sum_i I_1(\mathbf{x}_i + \mathbf{u}) \quad (7.13)$$

are the *mean images* of the corresponding patches and N is the number of pixels in the patch. The normalized cross-correlation score is always guaranteed to be in the range $[-1, 1]$, which makes it easier to handle in some higher-level applications (such as deciding which patches truly match). Note, however, that the NCC score is undefined if either of the two patches has zero variance (and in fact, its performance degrades for noisy low-contrast regions). [Note: For this reason, it may be preferable to use Bayesian priors on the bias and gain parameters (A. Criminisi (2004), personal communication, 9/22/2004, notes saved in Antonios_RNCC.ps). Replace above with a proper citation, e.g., cite{Criminisi04q}. Also, say something about the Normalized Sum of Squared Differences from (Criminisi et al. 2003), which may be most useful when evaluating a large number of overlapped windows such as in stereo matching - in stereo section of book?]

7.1.1 Hierarchical motion estimation

Now that you have a well-defined alignment cost function to optimize, how can you find its minimum? The simplest solution is to do a *full search* over some range of shifts, using either integer or sub-pixel steps. This is often the approach used for *block matching* in *motion compensated video compression*, where a range of possible motions (say ± 16 pixels) is explored.²

To accelerate this search process, *hierarchical motion estimation* is often used, where an image pyramid is first constructed, and a search over a smaller number of discrete pixels (corresponding to the same range of motion) is first performed at coarser levels (Quam 1984, Anandan 1989, Bergen et al. 1992). The motion estimate from one level of the pyramid can then be used to initialize a smaller *local* search at the next finer level. While this is not guaranteed to produce the same result as full search, it usually works almost as well and is much faster.

More formally, let

$$I_k^{(l)}(\mathbf{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\mathbf{x}_j) \quad (7.14)$$

be the *decimated* image at level l obtained by subsampling (*downsampling*) a smoothed (pre-filtered) version of the image at level $l-1$. (See §3.4 on how to perform the required downsampling (pyramid construction) without introducing aliasing.)

² In stereo matching §10.1.2, an explicit search over all possible disparities (i.e., a *plane sweep*) is almost always performed, since the number of search hypotheses is much smaller due to the 1D nature of the potential displacements.

At the coarsest level, we search for the best displacement $\mathbf{u}^{(l)}$ that minimizes the difference between images $I_0^{(l)}$ and $I_1^{(l)}$. This is usually done using a full search over some range of displacements $\mathbf{u}^{(l)} \in 2^{-l}[-S, S]^2$, where S is the desired *search range* at the finest (original) resolution level, optionally followed by the incremental refinement step described in §7.1.3.

Once a suitable motion vector has been estimated, it is used to *predict* a likely displacement

$$\hat{\mathbf{u}}^{(l-1)} \leftarrow 2\mathbf{u}^{(l)} \quad (7.15)$$

for the next finer level.³ The search over displacements is then repeated at the finer level over a much narrower range of displacements, say $\hat{\mathbf{u}}^{(l-1)} \pm 1$, again optionally combined with an incremental refinement step (Anandan 1989). A nice description of the whole process, extended to parametric motion estimation (§7.2), can be found in (Bergen *et al.* 1992).

7.1.2 Fourier-based alignment

When the search range corresponds to a significant fraction of the larger image (as is the case in image stitching), the hierarchical approach may not work that well, since it is often not possible to coarsen the representation too much before significant features get blurred away. In this case, a Fourier-based approach may be preferable.

Fourier-based alignment relies on the fact that the Fourier transform of a shifted signal has the same magnitude as the original signal but linearly varying phase §3.3, i.e.,

$$\mathcal{F}\{I_1(\mathbf{x} + \mathbf{u})\} = \mathcal{F}\{I_1(\mathbf{x})\} e^{-2\pi j \mathbf{u} \cdot \mathbf{f}} = \mathcal{I}_1(\mathbf{f}) e^{-2\pi j \mathbf{u} \cdot \mathbf{f}}, \quad (7.16)$$

where \mathbf{f} is the vector-valued frequency of the Fourier transform and we use calligraphic notation $\mathcal{I}_1(\mathbf{f}) = \mathcal{F}\{I_1(\mathbf{x})\}$ to denote the Fourier transform of a signal §3.3.

Another useful property of Fourier transforms is that convolution in the spatial domain corresponds to multiplication in the Fourier domain §3.3.⁴ Thus, the Fourier transform of the cross-correlation function E_{CC} can be written as

$$\mathcal{F}\{E_{CC}(\mathbf{u})\} = \mathcal{F}\left\{\sum_i I_0(\mathbf{x}_i)I_1(\mathbf{x}_i + \mathbf{u})\right\} = \mathcal{F}\{I_0(\mathbf{u})\bar{*}I_1(\mathbf{u})\} = \mathcal{I}_0(\mathbf{f})\mathcal{I}_1^*(\mathbf{f}), \quad (7.17)$$

where

$$f(\mathbf{u})\bar{*}g(\mathbf{u}) = \sum_i f(\mathbf{x}_i)g(\mathbf{x}_i + \mathbf{u}) \quad (7.18)$$

³ This doubling of displacements is only necessary if displacements are defined in integer *pixel* coordinates, which is the usual case in the literature, e.g., (Bergen *et al.* 1992). If *normalized device coordinates* (§2.1.4) are used instead, the displacements (and search ranges) need not change from level to level, although the step sizes will need to be adjusted (to keep search steps of roughly one pixel).

⁴ In fact, the Fourier shift property (7.16) derives from the convolution theorem by observing that shifting is equivalent to convolution with a displaced delta function $\delta(\mathbf{x} - \mathbf{u})$.

is the *correlation* function, i.e., the convolution of one signal with the reverse of the other, and $\mathcal{I}_1^*(\mathbf{f})$ is the *complex conjugate* of $\mathcal{I}_1(\mathbf{f})$. (This is because convolution is defined as the summation of one signal with the reverse of the other §3.3.)

Thus, to efficiently evaluate E_{CC} over the range of all possible values of \mathbf{u} , we take the Fourier transforms of both images $I_0(\mathbf{x})$ and $I_1(\mathbf{x})$, multiply both transforms together (after conjugating the second one), and take the inverse transform of the result. The Fast Fourier Transform algorithm can compute the transform of an $N \times M$ image in $O(NM \log NM)$ operations (Bracewell 1986). This can be significantly faster than the $O(N^2M^2)$ operations required to do a full search when the full range of image overlaps is considered.

While Fourier-based convolution is often used to accelerate the computation of image correlations, it can also be used to accelerate the sum of squared differences function (and its variants) as well. Consider the SSD formula given in (7.1). Its Fourier transform can be written as

$$\mathcal{F}\{E_{SSD}(\mathbf{u})\} = \mathcal{F}\left\{\sum_i [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2\right\} = \delta(\mathbf{f}) \sum_i [I_0^2(\mathbf{x}_i) + I_1^2(\mathbf{x}_i)] - 2\mathcal{I}_0(\mathbf{f})\mathcal{I}_1^*(\mathbf{f}). \quad (7.19)$$

Thus, the SSD function can be computed by taking twice the correlation function and subtracting it from the sum of the energies in the two images.

Windowed correlation. Unfortunately, the Fourier convolution theorem only applies when the summation over \mathbf{x}_i is performed over *all* the pixels in both images, using a circular shift of the image when accessing pixels outside the original boundaries. While this is acceptable for small shifts and comparably sized images, it makes no sense when the images overlap by a small amount or one image is a small subset of the other.

In that case, the cross-correlation function should be replaced with a *windowed* (weighted) cross-correlation function,

$$E_{WCC}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)I_0(\mathbf{x}_i) w_1(\mathbf{x}_i + \mathbf{u})I_1(\mathbf{x}_i + \mathbf{u}), \quad (7.20)$$

$$= [w_0(\mathbf{x})I_0(\mathbf{x})] \bar{*} [w_1(\mathbf{x})I_1(\mathbf{x})] \quad (7.21)$$

where the weighting functions w_0 and w_1 are zero outside the valid ranges of the images, and both images are padded so that circular shifts return 0 values outside the original image boundaries.

An even more interesting case is the computation of the *weighted* SSD function introduced in (7.5),

$$E_{WSSD}(\mathbf{u}) = \sum_i w_0(\mathbf{x}_i)w_1(\mathbf{x}_i + \mathbf{u})[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2. \quad (7.22)$$

Expanding this as a sum of correlations and deriving the appropriate set of Fourier transforms is left as an exercise (Exercise 7.1).

[Note: Generate the polar and log-polar images in MatLab?] (a) (b) (c)

Figure 7.2: An image (a) and its polar (b) and log-polar (c) transforms.

[Note: Show scaled and rotated versions as well, to see how they shift.] [Note: Need to write the code to generate these...]

The same kind of derivation can also be applied to the bias-gain corrected sum of squared difference function E_{BG} . Again, Fourier transforms can be used to efficiently compute all the correlations needed to perform the linear regression in the bias and gain parameters in order to estimate the exposure-compensated difference for each potential shift.

Phase correlation. A variant of regular correlation (7.17) that is sometimes used for motion estimation is *phase correlation* (Kuglin and Hines 1975, Brown 1992). Here, the spectrum of the two signals being matched is *whitened* by dividing each per-frequency product in (7.17) by the magnitudes of the Fourier transforms,

$$\mathcal{F}\{E_{\text{PC}}(\mathbf{u})\} = \frac{\mathcal{I}_0(\mathbf{f})\mathcal{I}_1^*(\mathbf{f})}{\|\mathcal{I}_0(\mathbf{f})\| \|\mathcal{I}_1(\mathbf{f})\|} \quad (7.23)$$

before taking the final inverse Fourier transform. In the case of noiseless signals with perfect (cyclic) shift, we have $I_1(\mathbf{x} + \mathbf{u}) = I_0(\mathbf{x})$, and hence from (7.16) we obtain

$$\begin{aligned} \mathcal{F}\{I_1(\mathbf{x} + \mathbf{u})\} &= \mathcal{I}_1(\mathbf{f})e^{-2\pi j\mathbf{u}\cdot\mathbf{f}} = \mathcal{I}_0(\mathbf{f}) \text{ and} \\ \mathcal{F}\{E_{\text{PC}}(\mathbf{u})\} &= e^{-2\pi j\mathbf{u}\cdot\mathbf{f}}. \end{aligned} \quad (7.24)$$

The output of phase correlation (under ideal conditions) is therefore a single spike (impulse) located at the correct value of \mathbf{u} , which (in principle) makes it easier to find the correct estimate.

Phase correlation has a reputation in some quarters of outperforming regular correlation, but this behavior depends on the characteristics of the signals and noise. If the original images are contaminated by noise in a narrow frequency band (e.g., low-frequency noise or peaked frequency “hum”), the whitening process effectively de-emphasizes the noise in these regions. However, if the original signals have very low signal-to-noise ratio at some frequencies (say, two blurry or low-textured images with lots of high-frequency noise), the whitening process can actually decrease performance (see Exercise 7.1).

Recently, gradient cross-correlation has emerged as a promising alternative to phase correlation (Argyriou and Vlachos 2003), although further systematic studies are probably warranted. Phase correlation has also been studied by Fleet and Jepson (1990) as a method for estimating general optical flow and stereo disparity.

Rotations and scale. [Note: This section can be shortened if pressed for space.] While Fourier-based alignment is mostly used to estimate translational shifts between images, it can, under certain limited conditions, also be used to estimate in-plane rotations and scales. Consider two images that are related *purely* by rotation, i.e.,

$$I_1(\hat{\mathbf{R}}\mathbf{x}) = I_0(\mathbf{x}). \quad (7.25)$$

If we re-sample the images into *polar coordinates* (Figure 7.2b),

$$\tilde{I}_0(r, \theta) = I_0(r \cos \theta, r \sin \theta) \text{ and } \tilde{I}_1(r, \theta) = I_1(r \cos \theta, r \sin \theta), \quad (7.26)$$

we obtain

$$\tilde{I}_1(r, \theta + \hat{\theta}) = \tilde{I}_0(r, \theta). \quad (7.27)$$

The desired rotation can then be estimated using an FFT shift-based technique.

If the two images are also related by a scale,

$$I_1(e^{\hat{s}} \hat{\mathbf{R}}\mathbf{x}) = I_0(\mathbf{x}), \quad (7.28)$$

we can re-sample into *log-polar coordinates* (Figure 7.2c),

$$\tilde{I}_0(s, \theta) = I_0(e^s \cos \theta, e^s \sin \theta) \text{ and } \tilde{I}_1(s, \theta) = I_1(e^s \cos \theta, e^s \sin \theta), \quad (7.29)$$

to obtain

$$\tilde{I}_1(s + \hat{s}, \theta + \hat{\theta}) = I_0(s, \theta). \quad (7.30)$$

In this case, care must be taken to choose a suitable range of s values that reasonably samples the original image.

For images that are also translated by a small amount,

$$I_1(e^{\hat{s}} \hat{\mathbf{R}}\mathbf{x} + \mathbf{t}) = I_0(\mathbf{x}), \quad (7.31)$$

De Castro and Morandi (1987) proposed an ingenious solution that uses several steps to estimate the unknown parameters. First, both images are converted to the Fourier domain, and only the magnitudes of the transformed images are retained. In principle, the Fourier magnitude images are insensitive to translations in the image plane (although the usual caveats about border effects apply). Next, the two magnitude images are aligned in rotation and scale using the polar or log-polar representations. Once rotation and scale are estimated, one of the images can be de-rotated and scaled, and a regular translational algorithm can be applied to estimate the translational shift.

Unfortunately, this trick only applies when the images have large overlap (small translational motion). For more general motion of patches or images, the parametric motion estimator described in §7.2 or the feature-based approaches described in §5.1 need to be used.

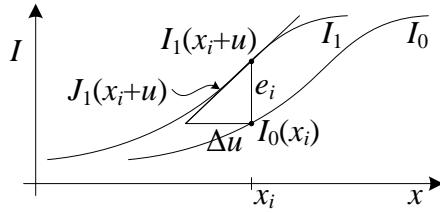


Figure 7.3: *Taylor series approximation of a function and the incremental computation of the optical flow correction amount.* $\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})$ is the image gradient at $(\mathbf{x}_i + \mathbf{u})$ and e_i is the current intensity difference.

7.1.3 Incremental refinement

The techniques described up till now can estimate translational alignment to the nearest pixel (or potentially fractional pixel if smaller search steps are used). In general, image stabilization and stitching applications require much higher accuracies to obtain acceptable results.

To obtain better *sub-pixel* estimates, we can use one of several techniques (Tian and Huhns 1986). One possibility is to evaluate several discrete (integer or fractional) values of (u, v) around the best value found so far and to *interpolate* the matching score to find an analytic minimum.

A more commonly used approach, first proposed by Lucas and Kanade (1981), is to do *gradient descent* on the SSD energy function (7.1), using a Taylor Series expansion of the image function (Figure 7.3),

$$E_{\text{LK-SSD}}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u} + \Delta\mathbf{u}) - I_0(\mathbf{x}_i)]^2 \quad (7.32)$$

$$\approx \sum_i [I_1(\mathbf{x}_i + \mathbf{u}) + \mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} - I_0(\mathbf{x}_i)]^2 \quad (7.33)$$

$$= \sum_i [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i]^2, \quad (7.34)$$

where

$$\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) = \nabla I_1(\mathbf{x}_i + \mathbf{u}) = \left(\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y} \right)(\mathbf{x}_i + \mathbf{u}) \quad (7.35)$$

is the *image gradient* at $(\mathbf{x}_i + \mathbf{u})$ and

$$e_i = I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i), \quad (7.36)$$

first introduced in (7.1), is the current intensity error.⁵ The gradient at a particular sub-pixel location $(\mathbf{x}_i + \mathbf{u})$ can be computed using a variety of techniques, the simplest of which is to simply

⁵ We follow the convention, commonly used in robotics and in (Baker and Matthews 2004), that derivatives with respect to (column) vectors result in row vectors, so that fewer transposes are needed in the formulas.

take the horizontal and vertical differences between pixels \mathbf{x} and $\mathbf{x} + (1, 0)$ or $\mathbf{x} + (0, 1)$. More sophisticated derivatives can sometimes lead to noticeable performance improvements. [Note: Find some references here. Does Simon Baker study this? Is it in (Tian and Huhns 1986) or Weickert's papers (Bruhn et al. 2005, Papenberg et al. 2006)? In feature detection, it's common to pre-filter the image with a small Gaussian (Förstner 1986, Harris and Stephens 1988).]

The above least squares problem can be minimizing by solving the associated *normal equations* (Golub and Van Loan 1996), [Note: Check if normal equations are in (Golub and Van Loan 1996); Aseem cites Meyer 2000 (*Matrix Analysis and Applied Linear Algebra*). Borrow it, and compare to Golub and VanLoan as well as Strang.]

$$\mathbf{A}\Delta\mathbf{u} = \mathbf{b} \quad (7.37)$$

where

$$\mathbf{A} = \sum_i \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u}) \mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \quad (7.38)$$

and

$$\mathbf{b} = - \sum_i e_i \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u}) \quad (7.39)$$

are called the (Gauss-Newton approximation of the) *Hessian* and *gradient-weighted residual vector*, respectively.⁶ These matrices are also often written as

$$\mathbf{A} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \text{ and } \mathbf{b} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}, \quad (7.40)$$

where the subscripts in I_x and I_y denote spatial derivatives, and I_t is called the *temporal derivative*, which makes sense if we are computing instantaneous velocity in a video sequence.

The gradients required for $\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})$ can be evaluated at the same time as the image warps required to estimate $I_1(\mathbf{x}_i + \mathbf{u})$, and in fact are often computed as a side-product of image interpolation. If efficiency is a concern, these gradients can be replaced by the gradients in the *template* image,

$$\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \approx \mathbf{J}_0(\mathbf{x}_i), \quad (7.41)$$

since near the correct alignment, the template and displaced target images should look similar. This has the advantage of allowing the pre-computation of the Hessian and Jacobian images, which can result in significant computational savings (Hager and Belhumeur 1998, Baker and Matthews 2004). A further reduction in computation can be obtained by writing the warped image $I_1(\mathbf{x}_i + \mathbf{u})$ used to compute e_i in (7.36) as a convolution of a sub-pixel interpolation filter with the discrete

⁶ The true Hessian is the full second derivative of the error function E , which may not be positive definite. [Note: Add reference to someplace where this Newton approximation is first discussed. In §5.1.3 or Appendix A.3.]

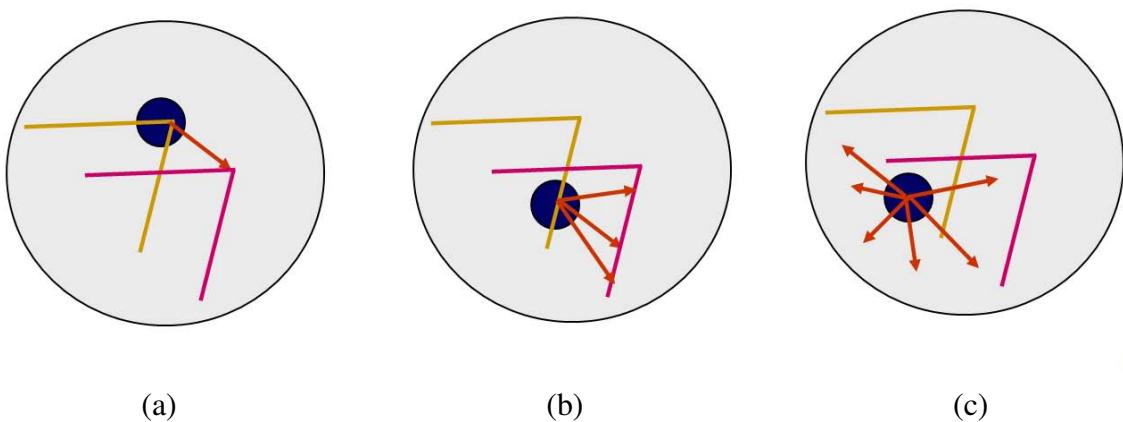


Figure 7.4: *Aperture problems for different image patches:* (a) stable (“corner-like”) flow; (b) classic aperture problem (barber-pole illusion); (c) textureless region. The two images I_0 (yellow) and I_1 (red) are overlaid. The red vector \mathbf{u} indicates the motion between the patch centers, and the $w(\mathbf{x}_i)$ window is shown as a dark circle.

samples in I_1 (Peleg and Rav-Acha 2006). Precomputing the inner product between the gradient field and shifted version of I_1 allows the iterative re-computation of e_i to be performed in constant time (independent of the number of pixels).

The effectiveness of the above incremental update rule relies on the quality of the Taylor series approximation. When far away from the true displacement (say 1-2 pixels), several iterations may be needed. (It is possible, however, to estimate a value for \mathbf{J}_1 using a least-squares fit to a series of larger displacements in order to increase the range of convergence (Jurie and Dhome 2002).) [Note: This is better for tracking, where it makes sense to spend this extra time to “learn” a better tracker. Ollie (Williams et al. 2003) also cites Shai Avidan’s “SVM tracker” (Avidan 2001).] When started in the vicinity of the correct solution, only a few iterations usually suffice. A commonly used stopping criterion is to monitor the magnitude of the displacement correction $\|\mathbf{u}\|$ and to stop when it drops below a certain threshold (say $1/10^{\text{th}}$ of a pixel). For larger motions, it is usual to combine the incremental update rule with a hierarchical coarse-to-fine search strategy, as described in §7.1.1.

Conditioning and aperture problems. Sometimes, the inversion of the linear system (7.37) can be poorly conditioned because of lack of two-dimensional texture in the patch being aligned. A commonly occurring example of this is the *aperture problem*, first identified in some of the early papers on optical flow (Horn and Schunck 1981) and then studied more extensively by Anandan (1989). Consider an image patch that consists of a slanted edge moving to the right (Figure 7.4). Only the *normal* component of the velocity (displacement) can be reliably recovered in this case.

This manifests itself in (7.37) as a *rank-deficient* matrix \mathbf{A} , i.e., one whose smaller eigenvalue is very close to zero.⁷

When equation (7.37) is solved, the component of the displacement along the edge is very poorly conditioned and can result in wild guesses under small noise perturbations. One way to mitigate this problem is to add a *prior* (soft constraint) on the expected range of motions (Simoncelli *et al.* 1991, Baker *et al.* 2004, Govindu 2006). This can be accomplished by adding a small value to the diagonal of \mathbf{A} , which essentially biases the solution towards smaller $\Delta\mathbf{u}$ values that still (mostly) minimize the squared error.

However, the pure Gaussian model assumed when using a simple (fixed) quadratic prior, as in (Simoncelli *et al.* 1991), does not always hold in practice, e.g., because of aliasing along strong edges (Triggs 2004). For this reason, it may be prudent to add some small fraction (say 5%) of the larger eigenvalue to the smaller one before doing the matrix inversion.

Uncertainty modeling. [Note: I've already covered some of this in §5.1.7 and Appendix B.7. Need to rationalize.] The reliability of a particular patch-based motion estimate can be captured more formally with an *uncertainty model*. The simplest such model is a *covariance matrix*, which captures the expected variance in the motion estimate in all possible directions. Under small amounts of additive Gaussian noise, it can be shown that the covariance matrix $\Sigma_{\mathbf{u}}$ is proportional to the inverse of the Hessian \mathbf{A} ,

$$\Sigma_{\mathbf{u}} = \sigma_n^2 \mathbf{A}^{-1}, \quad (7.42)$$

where σ_n^2 is the variance of the additive Gaussian noise (Anandan 1989, Matthies *et al.* 1989, Szeliski 1989). For larger amounts of noise, the linearization performed by the Lucas-Kanade algorithm in (7.34) is only approximate, so the above quantity becomes the *Cramer-Rao lower bound* on the true covariance. Thus, the minimum and maximum eigenvalues of the Hessian \mathbf{A} can now be interpreted as the (scaled) inverse variances in the least-certain and most-certain directions of motion. (A more detailed analysis using a more realistic model of image noise can be found in (Steele and Jaynes 2005).)

Bias and gain, weighting, and robust error metrics. The Lucas-Kanade update rule can also be applied to the bias-gain equation (7.9) to obtain

$$E_{\text{LK-BG}}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u})\Delta\mathbf{u} + e_i - \alpha I_0(\mathbf{x}_i) - \beta]^2 \quad (7.43)$$

(Lucas and Kanade 1981, Gennert 1988, Fuh and Maragos 1991, Baker *et al.* 2003b). The resulting 4×4 system of equations can be solved to simultaneously estimate the translational displacement

⁷The matrix \mathbf{A} is by construction always guaranteed to be symmetric positive semi-definite, i.e., it has real non-negative eigenvalues.

update $\Delta\mathbf{u}$ and the bias and gain parameters β and α .⁸

A similar formulation can be derived for images (templates) that have a *linear appearance variation*,

$$I_1(\mathbf{x} + \mathbf{u}) \approx I_0(\mathbf{x}) + \sum_j \lambda_j B_j(\mathbf{x}), \quad (7.44)$$

where the $B_j(\mathbf{x})$ are the *basis images* and the λ_j are the unknown coefficients (Hager and Belhumeur 1998, Baker *et al.* 2003a, Baker *et al.* 2003b). Potential linear appearance variations include illumination changes (Hager and Belhumeur 1998) and small non-rigid deformations (Black and Jepson 1998).

A weighted (windowed) version of the Lucas-Kanade algorithm is also possible,

$$E_{\text{LK-WSSD}}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i w_0(\mathbf{x}_i) w_1(\mathbf{x}_i + \mathbf{u}) [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \Delta\mathbf{u} + e_i]^2. \quad (7.45)$$

Note that here, in deriving the Lucas-Kanade update from the original weighted SSD function (7.5), we have neglected taking the derivative of $w_1(\mathbf{x}_i + \mathbf{u})$ weighting function with respect to \mathbf{u} , which is usually acceptable in practice, especially if the weighting function is a binary mask with relatively few transitions.

Baker *et al.* (2003a) only use the $w_0(\mathbf{x})$ term, which is reasonable if the two images have the same extent and no (independent) cutouts in the overlap region. They also discuss the idea of making the weighting proportional to $\nabla I(\mathbf{x})$, which helps for very noisy images, where the gradient itself is noisy. Similar observation, formulated in terms of *total least squares* (Van Huffel and Vandewalle 1991), have been made by other researchers studying optical flow (motion) estimation (Weber and Malik 1995, Bab-Hadiashar and Suter 1998b, Mühlich and Mester 1998). Lastly, Baker *et al.* (2003a) show how evaluating (7.45) at just the *most reliable* (highest gradient) pixels does not significantly reduce performance for large enough images, even if only 5%-10% of the pixels are used. (This idea was originally proposed by Dellaert and Collins (1999), who used a more sophisticated selection criterion.)

The Lucas-Kanade incremental refinement step can also be applied to the robust error metric introduced in §7.1,

$$E_{\text{LK-SRD}}(\mathbf{u} + \Delta\mathbf{u}) = \sum_i \rho(e_i) \mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \Delta\mathbf{u} + e_i. \quad (7.46)$$

We can take the derivative of this function w.r.t. \mathbf{u} and set it to 0,

$$\sum_i \psi(e_i) \frac{\partial e_i}{\partial \mathbf{u}} = \sum_i \psi(e_i) \mathbf{J}_1(\mathbf{x} + \mathbf{u}) = 0, \quad (7.47)$$

⁸ In practice, it may be possible to decouple the bias-gain and motion update parameters, i.e., to solve two independent 2×2 systems, which is a little faster.

where $\Psi(e) = \rho'(e)$ is the derivative of ρ . If we introduce a weight function $w(e) = \Psi(e)/e$, we can write this as

$$\sum_i w(e_i) \mathbf{J}_1^T(\mathbf{x}_i + \mathbf{u}) [\mathbf{J}_1(\mathbf{x}_i + \mathbf{u}) \Delta \mathbf{u} + e_i] = 0. \quad (7.48)$$

This results in the *Iteratively Re-weighted Least Squares* algorithm, which alternates between computing the weight functions $w(e_i)$ and solving the above weighted least squares problem (Huber 1981, Stewart 1999). Alternative incremental robust least squares algorithms can be found in (Sawhney and Ayer 1996, Black and Anandan 1996, Black and Rangarajan 1996, Baker *et al.* 2003a) and textbooks and tutorials on robust statistics (Huber 1981, Hampel *et al.* 1986, Rousseeuw and Leroy 1987, Stewart 1999).

7.2 Parametric motion

Many image alignment tasks, for example image stitching with handheld cameras, require the use of more sophisticated motion models, as described in §2.1.2. Since these models typically have more parameters than pure translation, a full search over the possible range of values is impractical. Instead, the incremental Lucas-Kanade algorithm can be generalized to parametric motion models and used in conjunction with a hierarchical search algorithm (Lucas and Kanade 1981, Rehg and Witkin 1991, Fuh and Maragos 1991, Bergen *et al.* 1992, Shashua and Toelg 1997, Shashua and Wexler 2001, Baker and Matthews 2004). [Note: Check Simon's latest IJCV articles on his Web page and on IJCV Web site.]

For parametric motion, instead of using a single constant translation vector \mathbf{u} , we use a spatially varying *motion field* or *correspondence map*, $\mathbf{x}'(\mathbf{x}; \mathbf{p})$, parameterized by a low-dimensional vector \mathbf{p} , where \mathbf{x}' can be any of the motion models presented in §2.1.2. The parametric incremental motion update rule now becomes

$$E_{\text{LK-PM}}(\mathbf{p} + \Delta \mathbf{p}) = \sum_i [I_1(\mathbf{x}'(\mathbf{x}_i; \mathbf{p} + \Delta \mathbf{p})) - I_0(\mathbf{x}_i)]^2 \quad (7.49)$$

$$\approx \sum_i [I_1(\mathbf{x}'_i) + \mathbf{J}_1(\mathbf{x}'_i) \Delta \mathbf{p} - I_0(\mathbf{x}_i)]^2 \quad (7.50)$$

$$= \sum_i [\mathbf{J}_1(\mathbf{x}'_i) \Delta \mathbf{p} + e_i]^2, \quad (7.51)$$

where the Jacobian is now

$$\mathbf{J}_1(\mathbf{x}'_i) = \frac{\partial I_1}{\partial \mathbf{p}} = \nabla I_1(\mathbf{x}'_i) \frac{\partial \mathbf{x}'}{\partial \mathbf{p}}(\mathbf{x}_i), \quad (7.52)$$

i.e., the product of the image gradient ∇I_1 with the Jacobian of correspondence field, $\mathbf{J}_{x'} = \partial \mathbf{x}' / \partial \mathbf{p}$.

Transform	Matrix	Parameters	Jacobian $\mathbf{J}_{\mathbf{x}'}$
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	(h_{00}, \dots, h_{21})	(see text)

Table 7.1: Jacobians of the 2D coordinate transformations.

[Note: The following Table and explanation should be moved to 5.1.1.]

Table 7.1 shows the motion Jacobians $\mathbf{J}_{\mathbf{x}'}$ for the 2D planar transformations introduced in §2.1.2.⁹ Note how I have re-parameterized the motion matrices so that they are always the identity at the origin $\mathbf{p} = 0$. This will become useful below, when I talk about the compositional and inverse compositional algorithms. (It also makes it easier to impose priors on the motions.)

The derivatives in Table 7.1 are all fairly straightforward, except for the projective 2D motion (homography), which requires a per-pixel division to evaluate, c.f. (2.21), re-written here in its new parametric form as

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \text{ and } y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}. \quad (7.53)$$

The Jacobian is therefore

$$\mathbf{J}_{\mathbf{x}'} = \frac{\partial \mathbf{x}'}{\partial \mathbf{p}} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}, \quad (7.54)$$

where D is the denominator in (7.53), which depends on the current parameter settings (as do x' and y').

⁹ The derivatives of the 3D rotational motion model introduced in §8.1.3 are given in §5.1.6. Note that 3D rotational motion may not be introduced until §8.1.3, so we may not need this footnote.

For parametric motion, the (Gauss-Newton) *Hessian* and *gradient-weighted residual vector* become

$$\mathbf{A} = \sum_i \mathbf{J}_{\mathbf{x}'}^T(\mathbf{x}_i) [\nabla I_1^T(\mathbf{x}'_i) \nabla I_1(\mathbf{x}'_i)] \mathbf{J}_{\mathbf{x}'}(\mathbf{x}_i) \quad (7.55)$$

and

$$\mathbf{b} = - \sum_i \mathbf{J}_{\mathbf{x}'}^T(\mathbf{x}_i) [e_i \nabla I_1^T(\mathbf{x}'_i)]. \quad (7.56)$$

Note how the expressions inside the square brackets are the same ones evaluated for the simpler translational motion case (7.38–7.39).

Patch-based approximation. The computation of the Hessian and residual vectors for parametric motion can be significantly more expensive than for the translational case. For parametric motion with n parameters and N pixels, the accumulation of \mathbf{A} and \mathbf{b} takes $O(n^2N)$ operations (Baker and Matthews 2004). One way to reduce this by a significant amount is to divide the image up into smaller sub-blocks (patches) P_j and to only accumulate the simpler 2×2 quantities inside the square brackets at the pixel level (Shum and Szeliski 2000),

$$\mathbf{A}_j = \sum_{i \in P_j} \nabla I_1^T(\mathbf{x}'_i) \nabla I_1(\mathbf{x}'_i) \quad (7.57)$$

$$\mathbf{b}_j = \sum_{i \in P_j} e_i \nabla I_1^T(\mathbf{x}'_i). \quad (7.58)$$

The full Hessian and residual can then be approximated as

$$\mathbf{A} \approx \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) [\sum_{i \in P_j} \nabla I_1^T(\mathbf{x}'_i) \nabla I_1(\mathbf{x}'_i)] \mathbf{J}_{\mathbf{x}'}(\hat{\mathbf{x}}_j) = \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) \mathbf{A}_j \mathbf{J}_{\mathbf{x}'}(\hat{\mathbf{x}}_j) \quad (7.59)$$

and

$$\mathbf{b} \approx - \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) [\sum_{i \in P_j} e_i \nabla I_1^T(\mathbf{x}'_i)] = - \sum_j \mathbf{J}_{\mathbf{x}'}^T(\hat{\mathbf{x}}_j) \mathbf{b}_j, \quad (7.60)$$

where $\hat{\mathbf{x}}_j$ is the *center* of each patch P_j (Shum and Szeliski 2000). This is equivalent to replacing the true motion Jacobian with a piecewise-constant approximation. In practice, this works quite well. The relationship of this approximation to feature-based registration is discussed in §8.2.5.

Compositional approach. For a complex parametric motion such as a homography, the computation of the motion Jacobian becomes complicated, and may involve a per-pixel division. Szeliski and Shum (1997) observed that this can be simplified by first warping the target image I_1 according to the current motion estimate $\mathbf{x}'(\mathbf{x}; \mathbf{p})$,

$$\tilde{I}_1(\mathbf{x}) = I_1(\mathbf{x}'(\mathbf{x}; \mathbf{p})), \quad (7.61)$$

and then comparing this *warped* image against the template $I_0(\mathbf{x})$,

$$E_{\text{LK-SS}}(\Delta \mathbf{p}) = \sum_i [\tilde{I}_1(\tilde{\mathbf{x}}(\mathbf{x}_i; \Delta \mathbf{p})) - I_0(\mathbf{x}_i)]^2 \quad (7.62)$$

$$\approx \sum_i [\tilde{J}_1(\mathbf{x}_i) \Delta \mathbf{p} + e_i]^2 \quad (7.63)$$

$$= \sum_i [\nabla \tilde{I}_1(\mathbf{x}_i) \mathbf{J}_{\tilde{\mathbf{x}}}(\mathbf{x}_i) \Delta \mathbf{p} + e_i]^2. \quad (7.64)$$

Note that since the two images are assumed to be fairly similar, only an *incremental* parametric motion is required, i.e., the incremental motion can be evaluated around $\mathbf{p} = 0$, which can lead to considerable simplifications. For example, the Jacobian of the planar projective transform (7.53) now becomes

$$\mathbf{J}_{\tilde{\mathbf{x}}} = \left. \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{p}} \right|_{\mathbf{p}=0} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}. \quad (7.65)$$

Once the incremental motion $\tilde{\mathbf{x}}$ has been computed, it can be *prepended* to the previously estimated motion, which is easy to do for motions represented with transformation matrices, such as those given in Tables 2.1 and 7.1. Baker and Matthews (2004) call this the *forward compositional* algorithm, since the target image is being re-warped, and the final motion estimates are being composed.

If the appearance of the warped and template images is similar enough, we can replace the gradient of $\tilde{I}_1(\mathbf{x})$ with the gradient of $I_0(\mathbf{x})$, as suggested previously in (7.41). This has potentially a big advantage in that it allows the pre-computation (and inversion) of the Hessian matrix \mathbf{A} given in (7.55). The residual vector \mathbf{b} (7.56) can also be partially precomputed, i.e., the *steepest descent* images $\nabla I_0(\mathbf{x}) \mathbf{J}_{\tilde{\mathbf{x}}}(\mathbf{x})$ can be precomputed and stored for later multiplication with the $e(\mathbf{x}) = \tilde{I}_1(\mathbf{x}) - I_0(\mathbf{x})$ error images (Baker and Matthews 2004). This idea was first suggested by Hager and Belhumeur (1998) in what Baker and Matthews (2004) call a *forward additive* scheme.

Baker and Matthews (2004) introduce one more variant they call the *inverse compositional* algorithm. Rather than (conceptually) re-warping the warped target image $\tilde{I}_1(\mathbf{x})$, they instead warp the template image $I_0(\mathbf{x})$ and minimize

$$E_{\text{LK-BM}}(\Delta \mathbf{p}) = \sum_i [\tilde{I}_1(\mathbf{x}_i) - I_0(\tilde{\mathbf{x}}(\mathbf{x}_i; \Delta \mathbf{p}))]^2 \quad (7.66)$$

$$\approx \sum_i [\nabla I_0(\mathbf{x}_i) \mathbf{J}_{\tilde{\mathbf{x}}}(\mathbf{x}_i) \Delta \mathbf{p} - e_i]^2. \quad (7.67)$$

This is identical to the forward warped algorithm (7.64) with the gradients $\nabla \tilde{I}_1(\mathbf{x})$ replaced by the gradients $\nabla I_0(\mathbf{x})$, except for the sign of e_i . The resulting update $\Delta \mathbf{p}$ is the *negative* of the one computed by the modified (7.64), and hence the *inverse* of the incremental transformation must be prepended to the current transform. Because the inverse compositional algorithm has the

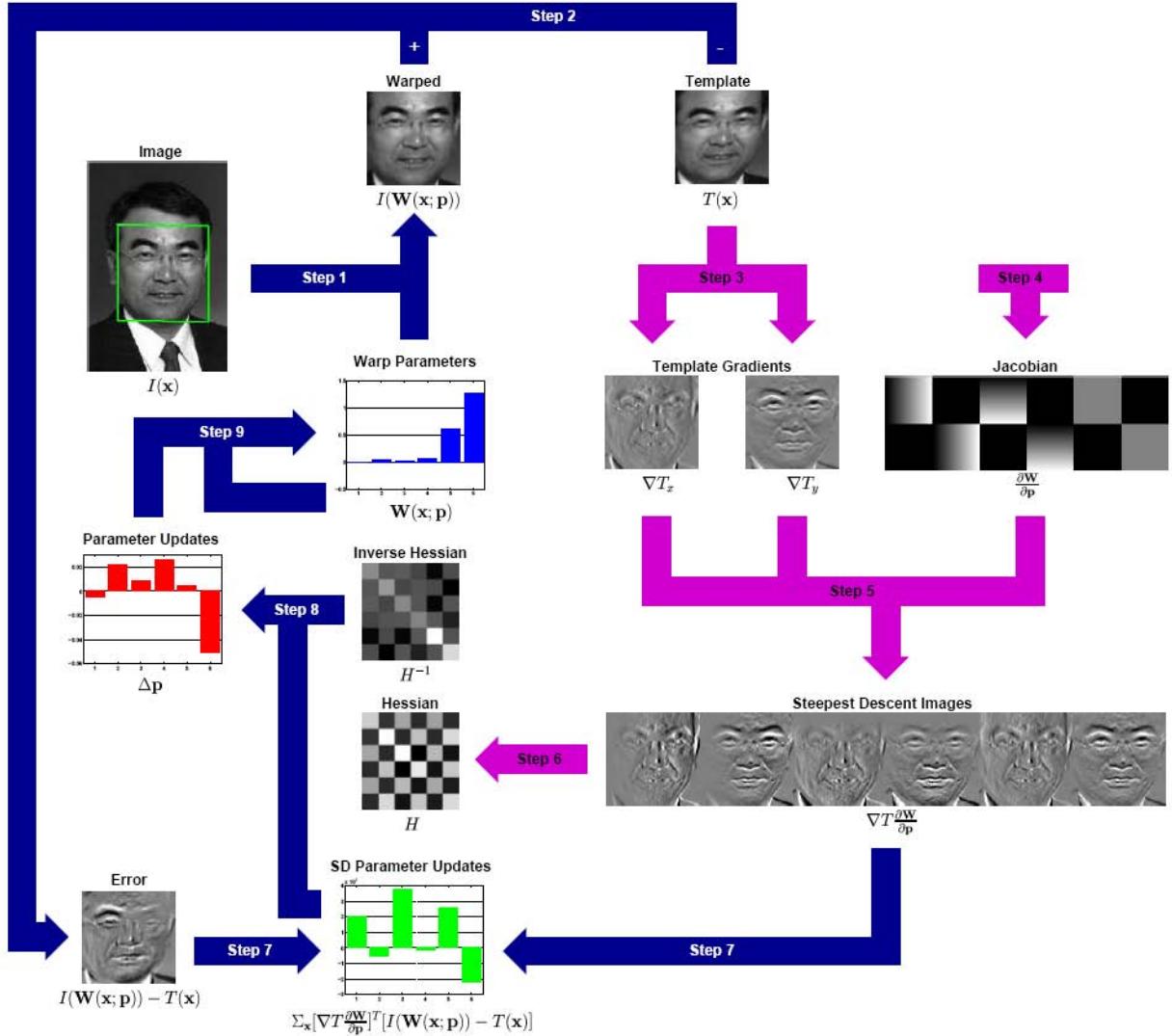


Figure 7.5: A schematic overview of the inverse compositional algorithm (copied, with permission, from (Baker et al. 2003a)). Steps 3-6 (light-color arrows) are performed once as a pre-computation. The main algorithm simply consists of iterating: image warping (Step 1), image differencing (Step 2), image dot products (Step 7), multiplication with the inverse of the Hessian (Step 8), and the update to the warp (Step 9). All of these steps can be performed efficiently.

potential of pre-computing the inverse Hessian and the steepest descent images, this makes it the preferred approach of those surveyed in (Baker and Matthews 2004). Figure 7.5, taken from (Baker *et al.* 2003a), beautifully shows all of the steps required to implement the inverse compositional algorithm. [*Note: If I want to save some space, the preceding 3 paragraphs can be compressed somewhat by dropping the equations and some of the text, as well as Figure 7.5. I've made a start at the compressed text below—uncomment the LaTeX source to see this text.*]

Baker and Matthews (2004) also discusses the advantage of using Gauss-Newton iteration (i.e., the first order expansion of the least squares, as above) vs. other approaches such as steepest descent and Levenberg-Marquardt. Subsequent parts of the series (Baker *et al.* 2003a, Baker *et al.* 2003b, Baker *et al.* 2004) discuss more advanced topics such as per-pixel weighting, pixel selection for efficiency, a more in-depth discussion of robust metrics and algorithms, linear appearance variations, and priors on parameters. They make for invaluable reading for anyone interested in implementing a highly tuned implementation of incremental image registration.

7.2.1 Application: Video stabilization

[*Note: Still need to write this section.*]

See (Matsushita *et al.* 2006) for a recent paper with in-painting, (Hansen *et al.* 1994, Irani *et al.* 1997, Morimoto and Chellappa 1997, Srinivasan *et al.* 2005) older work (the last is a survey, have a look).

Describe the steps:

1. translation (and rotation?) motion with robust outlier rejection (or multiple motion estimation and mode finding)
2. temporal filtering high-pass to remove low-frequency component
3. compensation by remaining motion, with cropping (zooming) in or fill-in from other frames

Forward pointer to video processing §13.1. (Also, given as an Exercise.)

7.3 Spline-based motion

While parametric motion models are useful in a wide variety of applications (such as video stabilization and mapping onto planar surfaces), most image motion is too complicated to be captured by such low-dimensional models.

Traditionally, optical flow algorithms have computed an independent motion estimate for each pixel, i.e., the number of flow vectors computed is equal to the number of input pixels (§7.4). The

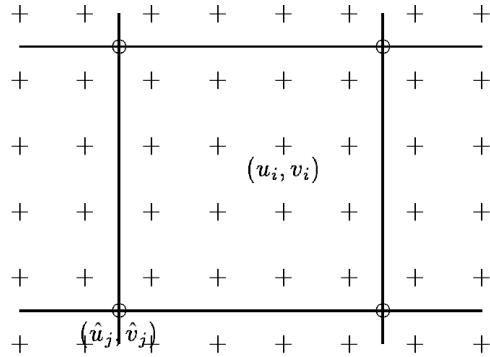


Figure 7.6: *Spline motion field: the displacement vectors $\mathbf{u}_i = (u_i, v_i)$ are shown as pluses (+) and are controlled by the smaller number of control vertices $\hat{\mathbf{u}}_j = (\hat{u}_j, \hat{v}_j)$, which are shown as circles (o).*

general optical flow analogue to (7.1) can thus be written as

$$E_{\text{SSDG}}(\{\mathbf{u}_i\}) = \sum_i [I_1(\mathbf{x}_i + \mathbf{u}_i) - I_0(\mathbf{x}_i)]^2. \quad (7.68)$$

Notice how in the above equation, the number of variables $\{\mathbf{u}_i\}$ is twice the number of measurements, so the problem is underconstrained.

The two classic approaches to this problem, which we will study in the next section §7.4, are to either to perform the summation over overlapping regions (the *patch-based* or *window-based* approach), or to add additional smoothness terms on the $\{\mathbf{u}_i\}$ field using *regularization* or *Markov random fields*.

In this section, I present an alternative approach that lies somewhere between general optical flow (independent flow at each pixel) and parametric flow (small number of global parameters). The approach is to represent the motion field as a two-dimensional *spline* controlled by a smaller number of *control vertices* $\{\hat{\mathbf{u}}_j\}$ (Figure 7.6),

$$\mathbf{u}_i = \sum_j \hat{\mathbf{u}}_j B_j(\mathbf{x}_i) == \sum_j \hat{\mathbf{u}}_j w_{i,j}, \quad (7.69)$$

where the $B_j(\mathbf{x}_i)$ are called the *basis functions* and are only non-zero over a small *finite support* interval (Szeliski and Coughlan 1997). We call the $w_{i,j} = B_j(\mathbf{x}_i)$ *weights* to emphasize that the $\{\mathbf{u}_i\}$ are known linear combinations of the $\{\hat{\mathbf{u}}_j\}$.

In our current implementation...

Can use higher-order models (Shashua and Toelg 1997, Shashua and Wexler 2001), but splines are better for more general motion (Szeliski and Coughlan 1997) (Szeliski and Shum 1996) (Szeliski and Kang 1995) (Kang *et al.* 1997) (Kybic and Unser 2003)—read [*Note: look at other Unser papers or book?*]

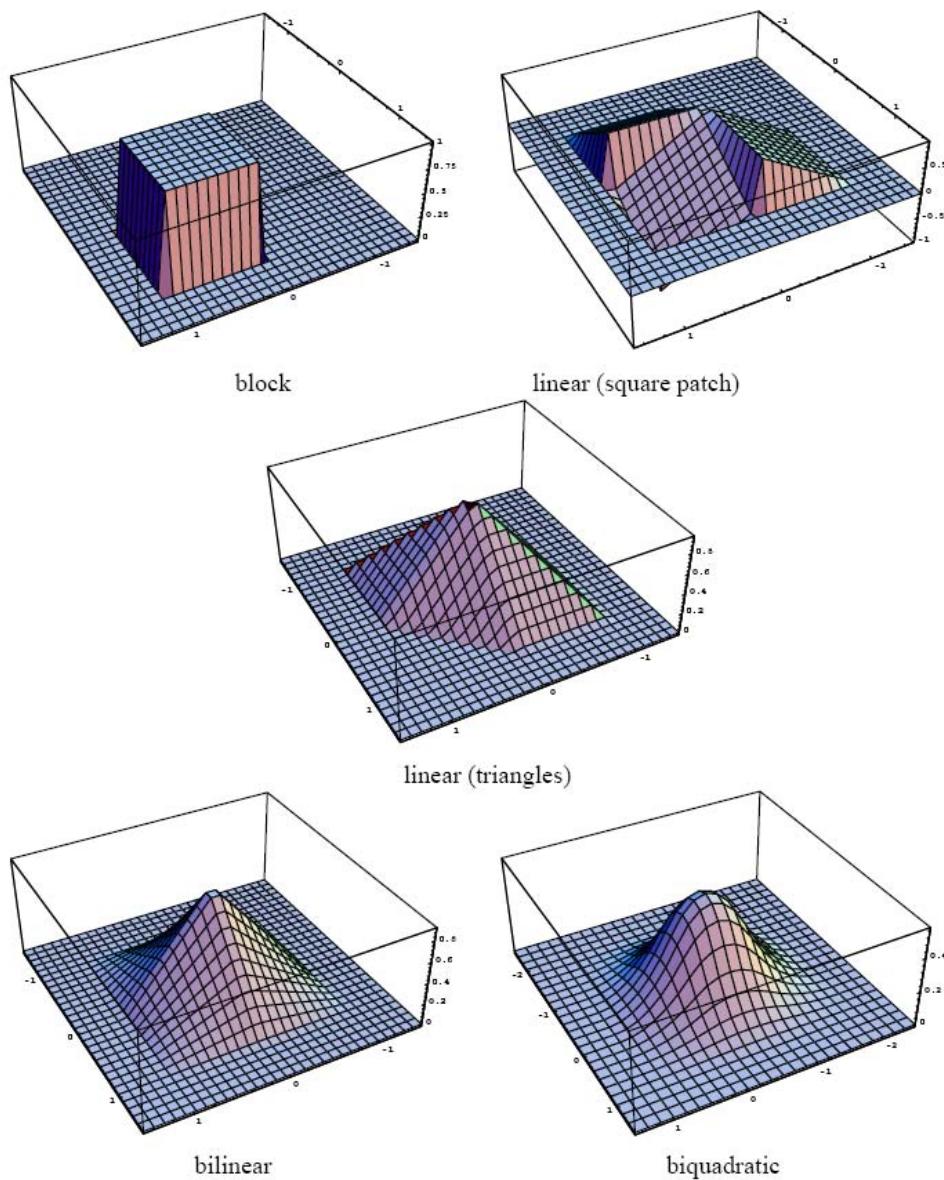


Figure 7.7: Spline basis function

[Note: This figure could also be moved to Image Processing §3 but it's probably fine here.]

Plane+parallax support ((Szeliski and Kang 1995, Szeliski and Coughlan 1997)
 Smoothness constraints (membrane and thin-plate)
 Solution with LDU decomposition (Appendix A.4).
 Solution with conjugate gradients, PCG, and hierarchical bases (Szeliski 1990b, Pentland 1994) (Appendix A.5).

7.3.1 Application: Automated morphing

Describe feature-based morphing (Beier and Neely 1992)

Describe automated flow-based morph (Beymer 1996) [Note: Journal version?]
 Forward pointer to morph-based de-ghosting (§8.3.2 and (Shum and Szeliski 2000))
 Can also be used to do frame interpolation (but prefer to use optical flow, since motion tends to be small and need pixel accuracy).

7.4 Optical flow

[Note: Incorporate some text from our recent survey (Baker et al. 2007) and cite it.]

Independent estimate at *each* pixel
 Reduction of spline model to 1x1 patches: Horn & Schunck (Horn and Schunck 1981) (like Lucas-Kanade (Lucas and Kanade 1981), but use smoothness instead of patches) Solved with iterative techniques (§A.5). (May want to use sparse CG to solve §A.5.)
 Robust versions (Black and Anandan 1996, Bab-Hadiashar and Suter 1998a)
 Overlapped patches: (Bergen et al. 1992)
 Read (Mémin and Pérez 2002) and see what's in there...
 Special Issue On Multimedia Signal Processing, Part I [Scanning the Past]. Tekalp, A.M. Proceedings of the IEEE. Volume: 86 5 , Page(s): 751 -754
 (Bruhn et al. 2005, Papenberg et al. 2006, Govindu 2006, Nir et al. 2008); also, Odobez & Bouhoumy; NIPS 2006_0342.pdf (in proceedings???)
 See what else is in (Baker et al. 2007);
 Newest CVPR'08 papers use MRFs and discrete (combinatorial) optimization, just like in stereo.
 Roth & Black ECCV paper (Sun et al. 2008) uses steerable random fields and learning to learn the best regularizers
 Glocker et al. (2008) use a coarse-to-fine strategy with per-pixel 2D uncertainties used to guide the refinement and search.

Lempitsky *et al.* (2008b) use fusion moves (Lempitsky *et al.* 2007) over proposals generated from basic flow algorithms (Horn and Schunck 1981, Lucas and Kanade 1981) and shifted variants to obtain (currently best) results.

7.4.1 Spatio-temporal filtering

[Note: Should the other main techniques have their own subsection*s ?]

(Heeger 1988), lots more out of Granlund, (Jähne 1997)

Barron survey (Barron *et al.* 1994), more recent surveys (see beginning of this chapter)

Where do phase-based motion estimates (Fleet and Jepson 1990, Fleet *et al.* 1991, Fleet 1992) (redundant references?) belong? They did well in the original (Barron *et al.* 1994) survey, but probably do not do that well near edges...

7.4.2 Application: Frame interpolation

Describe the classic frame interpolation problem (see also Exercise 7.5).

Estimate pixel-accurate motion, using more than two frames. Estimate motion at fractional time steps.

7.5 Layered motion

Original layered motion paper (Wang and Adelson 1994, Weiss and Adelson 1996)

3D rigid flow (Baker *et al.* 1998, Torr *et al.* 2001)

An algorithm for simultaneous motion estimation and scene segmentation. Chang, M.M.; Sezan, M.I.; Tekalp, A.M. Acoustics, Speech, and Signal Processing, 1994. ICASSP-94. Volume: v , Page(s): V/221 -V/224 vol.5 .

Skin & bones (Ju *et al.* 1996)

What about motion segmentation? Large literature (Mubarak Shah?, recent papers are (Stein *et al.* 2007, Thayanathan *et al.* 2008)).

Super-resolved layered representation: (Schoenemann and Cremers 2008)

Transparent layers and reflections

Transparent motion: use literature review from transparentLayers.tex

Transparent layer recovery (Darrell and Pentland 1991, Black and Rangarajan 1996, Szeliski *et al.* 2000). Also, pointer to related stereo with reflections work (Swaminathan *et al.* 2002, Tsin *et al.* 2003, Criminisi *et al.* 2005).



Figure 7.8: Example of light reflecting off the transparent glass of a picture frame: (a) first image from input sequence; (b) dominant motion layer min-composite; (c) secondary motion residual layer max-composite; (d–e) final estimated picture and reflection layers (Szeliski et al. 2000).

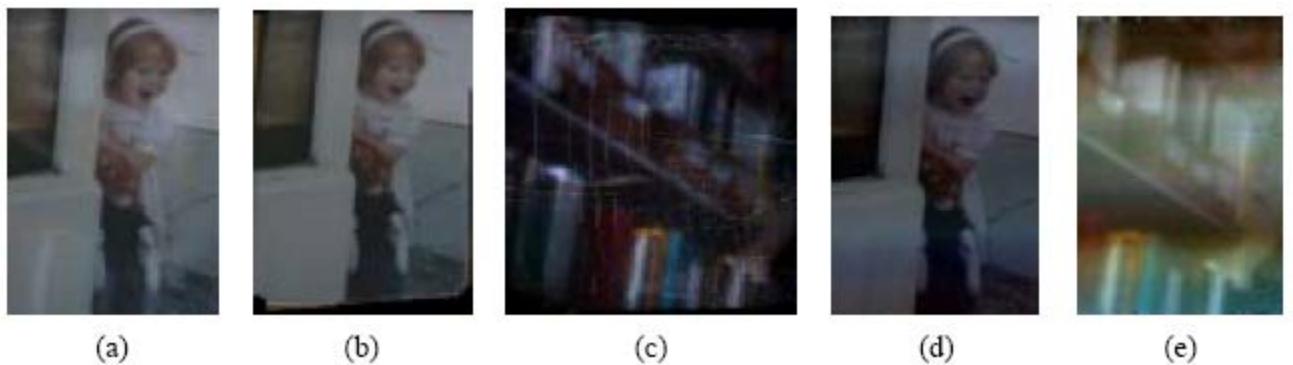


Figure 7.9: Another example of transparent motion separation: (a) first image from input sequence; (b) dominant motion layer min-composite; (c) secondary motion residual layer max-composite; (d–e) final estimated picture and reflection layers (Szeliski et al. 2000). Note that the reflected layers in (c) and (e) are doubled in intensity to better show their structure.

[Note: Make sure there's an exercise on this, since it's a fun project and one of the best ways to add realism to an IBR system. Also, see if the above stereo references can be added to the Stereo chapter.]

[Note: I also have transparent layers in §7.5 at the moment; need to rationalize and pick one place.]

Mention that image sequences with reflections are starting to appear in optical flow test sets (Baker *et al.* 2007).

More Black et al work?

(Kumar *et al.* 2008), based on earlier work by (Torr *et al.* 2001) and (Jovic and Frey 2001).

7.6 Learned motion models (tracking, revisited)

[Note: This topic suggested by Bill Triggs]

Black *et al.* (1997): is there a journal version? Torresani *et al.* (2008): look at references; Fua and Lepetit? (ECCV prize winner); Forsyth T-shirt tracking; Yasu's 3D tracking (Furukawa and Ponce 2008b); Fua group's non-rigid tracking (Salzmann *et al.* 2008).

[Note: Where do I really cover people tracking? In §11.7.3?]

7.6.1 Application: Motion-based user interface

Describe the Turk & Cutler motion-based user interface paper (Cutler and Turk 1998)

Also, decathlon running events; see Freeman's CG articles (Freeman *et al.* 1998, Freeman *et al.* 1999) and cites therein.

Another application of real-time tracking is *virtual mirrors*, first popularized by ... (MIT Media Lab) ... [6] T. Darrell, G. Gordon, J. Woodfill, and M. Harville. A virtual mirror interface using real-time robust face tracking. In Proc. Third International Conference on Face and Gesture Recognition, Nara, Japan, Apr. 1998?? ... Nice example is a virtual mirror for shoe buying (Eisert *et al.* 2008).

7.7 Exercises

[Note: Try to identify some open research problems for graduate students to work on...]

Ex 7.1 (Correlation (revisited)) Already implemented some simple one is Ex 4.3. Do I need to rationalize this?

Implement and compare the performance of the following correlation algorithms:

1. Sum of squared differences
2. Sum of absolute differences
3. Normalized cross-correlation
4. Windowed versions of the above (7.21–7.22)
5. Fourier-based implementations of the above measures (not the absolute differences one)
6. Phase correlation
7. Gradient cross-correlation (Argyriou and Vlachos 2003)

[Note: Provide some sample sequences] Discuss under what conditions each one is better.

Do a comparative study, with different kinds of noise and occlusion, different images (low-freq such as sand, textured cloth, ...). Ask when phase correlation might do better (low-frequency or “rolling” noise) or worse (blurred images, single frequency).

For the Fourier-based masked/windowed correlation and sum of squared differences, the results should be the same as the direct implementations. Note that you will have to expand (7.5) into a sum of pairwise correlations, just as in (7.21). (This is part of the exercise.)

Ex 7.2 (Affine registration) Implement direct method for affine and projective image alignment: different orders, coarse-to-fine (optional lower order at coarse level).

Optional: implement patch-based acceleration

See Baker’s recent survey (Baker and Matthews 2004) for more comparisons.

Ex 7.3 (Stabilization) Write a program to *stabilize* an input video sequence. You should implement the following steps, as described in §7.2.1:

1. Compute the translation (and optionally rotation) between successive frames with robust outlier rejection
2. Perform temporal high-pass filtering on the motion parameters to remove the low-frequency component (smooth motion).
3. Compensate for the high-frequency motion, zooming in slightly (user-specified amount) to avoid missing edge pixels.
4. (Optional) Do not zoom in, but instead borrow pixels from previous / subsequent frames to fill in.

[Note: Provide some sample sequences] Forward pointer to video processing §13.1?

Ex 7.4 (Optical flow) Compute optical flow (spline-based or per-pixel) between two images, using one or more of the techniques described in this chapter. *Or*, compute per-pixel flow, say using Bergen *et al.*'s hierarchical flow technique (maybe with spatio-temporal shiftable windows for better accuracy?)

Optional: compare the performance of your flow techniques.

Show the warped images as well as the flow images as a result.

Note: I currently do not have ground truth except for Yosemite and Otte and a few other synthetic sequences. Include these (after checking for permissions). But now, we have the ([Baker *et al.* 2007](#)) results: use these!

Ex 7.5 (Automated morphing / frame interpolation) Write a program to automatically morph between pairs of images. Implement the following steps, as described in §[7.3.1](#):

1. Compute the flow both ways (previous exercise), preferably at an intermediate point.
2. For each intermediate (morphed) image, warp each image partway towards its final appearance. [*Note: This can get tricky, since partial flow does not preserve features. What we really want is feature movement, i.e., forward warp. Use a 2-pass warper?*]
3. Blend (cross-dissolve) the images, and view with a sequence viewer (GIF image or simple app.) *or* Take a video sequence, and do a high-quality slow-mo (compare with frame repetition and frame dissolve).

[*Note: Provide some sample image pairs*] Try this out on images of your friends and colleagues.

Forward pointer to video processing §[13.1](#) for complete video morph (not just static pair, but full dynamic cross-dissolve as in Video Textures).

Ex 7.6 (Video de-noising) Implement the algorithm described in the application. [*Note: Provide some sample video*] Try this out on your old 8mm videos. (Also, need a pointer to the scratch removal literature...)

Forward pointer to video processing §[13.1](#) for complete video morph (not just static pair, but full dynamic cross-dissolve as in Video Textures). (Also reflect this in the text...)

Ex 7.7 (Motion-based user interaction—harder) Write a program to compute a low-resolution motion field in order to interactively control a simple application (§[7.6.1](#) and ([Cutler and Turk 1998](#))). For example:

1. Downsample each image using a pyramid, and compute the optical flow (spline-based or pixel-based) from the previous frame.

2. Segment each training video sequence into different “actions” (e.g., hand moving inwards, moving up, no motion, etc.), and “learn” the velocity fields associated with each one. (You can simply find the mean and variance for each motion field, or use something more sophisticated like a support vector machine (SVM).)
3. Write the recognizer that find successive actions of approximately the right duration, and hook it up to an interactive application (e.g., a sound generator or a computer game).
4. Test it out on your friends.

[Note: Provide some segmented/labeled sample video sequences]

Chapter 8

Image stitching

8.1	Motion models	370
8.1.1	Planar perspective motion	370
8.1.2	<i>Application:</i> Whiteboard and document scanning	372
8.1.3	Rotational panoramas	373
8.1.4	<i>Application:</i> Video summarization and compression	377
8.1.5	Cylindrical and spherical coordinates	378
8.2	Global alignment	382
8.2.1	Bundle adjustment	382
8.2.2	Parallax removal	386
8.2.3	Recognizing panoramas	387
8.2.4	<i>Application:</i> Full-view panoramas and virtual environments	388
8.2.5	Direct vs. feature-based alignment	388
8.3	Compositing	392
8.3.1	Choosing a compositing surface	392
8.3.2	Pixel selection and weighting (de-ghosting)	394
8.3.3	Blending	400
8.3.4	<i>Application:</i> Photomontage	403
8.4	Exercises	405



Figure 8.1: *Some examples of image stitching: (a) portion of a cylindrical panorama; (b) a spherical panorama constructed from 54 photographs; (c) a multi-image panorama automatically assembled from an unordered photo collection; (d–e) a multi-image stitch without and with moving object removal.*

Algorithms for aligning images and stitching them into seamless photo-mosaics are among the oldest and most widely used in computer vision (Milgram 1975, Peleg 1981). Image stitching algorithms create the high-resolution photo-mosaics used to produce today's digital maps and satellite photos. They also come bundled with most digital cameras currently being sold and can be used to create beautiful ultra wide-angle panoramas.

Image stitching originated in the photogrammetry community, where more manually intensive methods based on surveyed *ground control points* or manually registered *tie points* have long been used to register aerial photos into large-scale photo-mosaics (Slama 1980). One of the key advances in this community was the development of *bundle adjustment* algorithms (§6.4), which could simultaneously solve for the locations of all of the camera positions, thus yielding globally consistent solutions (Triggs *et al.* 1999). Another recurring problem in creating photo-mosaics is the elimination of visible seams, for which a variety of techniques have been developed over the years (Milgram 1975, Milgram 1977, Peleg 1981, Davis 1998, Agarwala *et al.* 2004).

In film photography, special cameras were developed at the turn of the century to take ultra wide angle panoramas, often by exposing the film through a vertical slit as the camera rotated on its axis (Meehan 1990). [Note: Add a figure/photo from (Meehan 1990) in the IBM book.] In the mid-1990s, image alignment techniques started being applied to the construction of wide-angle seamless panoramas from regular hand-held cameras (Mann and Picard 1994, Chen 1995, Szeliski 1996). More recent work in this area has addressed the need to compute globally consistent alignments (Szeliski and Shum 1997, Sawhney and Kumar 1999, Shum and Szeliski 2000), the removal of “ghosts” due to parallax and object movement (Davis 1998, Shum and Szeliski 2000, Uyttendaele *et al.* 2001, Agarwala *et al.* 2004), and dealing with varying exposures (Mann and Picard 1994, Uyttendaele *et al.* 2001, Levin *et al.* 2004, Agarwala *et al.* 2004, Eden *et al.* 2006, Kopf *et al.* 2007b). (A collection of some of these papers can be found in (Benosman and Kang 2001).) These techniques have spawned a large number of commercial stitching products (Chen 1995, Sawhney *et al.* 1998), for which reviews and comparison can be found on the Web.

While most of the earlier techniques worked by directly minimizing pixel-to-pixel dissimilarities, more recent algorithms usually extract a sparse set of *features* and then match these to each other, as described in Chapter §4. Such feature-based approaches to image stitching (Zoghliami *et al.* 1997, Capel and Zisserman 1998, Cham and Cipolla 1998, Badra *et al.* 1998, McLauchlan and Jaenicke 2002, Brown and Lowe 2003a) have the advantage of being more robust against scene movement and are potentially faster, if implemented the right way. Their biggest advantage, however, is the ability to “recognize panoramas”, i.e., to automatically discover the adjacency (overlap) relationships among an unordered set of images, which makes them ideally suited for fully automated stitching of panoramas taken by casual users (Brown and Lowe 2003a).

What, then, are the essential problems in image stitching? As with image alignment, we must first determine the appropriate mathematical model relating pixel coordinates in one image to pixel

coordinates in another. Section §8.1 reviews the basic models we have previously studied and presents some new motion models related specifically to panoramic image stitching. Next, we must somehow estimate the correct alignments relating various pairs (or collections) of images. Chapter §4 discussed how distinctive *features* can be found in each image and then efficiently matched to rapidly establish correspondences between pairs of images. Chapter §7 discussed how *direct* pixel-to-pixel comparisons combined with gradient descent (and other optimization techniques) can also be used to estimate these parameters. When multiple images exist in a panorama, bundle adjustment (§6.4) can be used to compute a globally consistent set of alignments and to efficiently discover which images overlap one another. In this chapter, we look at how each of these previously developed techniques can be modified to take advantage of the imaging setups commonly used to create panoramas.

Once we have aligned the images, we must choose a final compositing surface onto which to warp and place all of the aligned images (§ 8.3). We also need to develop algorithms to seamlessly blend overlapping images, even in the presence of parallax, lens distortion, scene motion, and exposure differences (§ 8.3). In the last section of this chapter, I discuss additional applications of image stitching and open research problems.

8.1 Motion models

Before we can register and align images, we need to establish the mathematical relationships that map pixel coordinates from one image to another. A variety of such *parametric motion models* are possible, from simple 2D transforms, to planar perspective models, 3D camera rotations, lens distortions, and the mapping to non-planar (e.g., cylindrical) surfaces.

I already covered a lot of these in the section on geometric image formation §2.1 and geometric alignment §5.1. In particular, we saw in §2.1.4 how the parametric motion describing the deformation of a planar surface as viewed from different positions can be described with an 8-parameter homography (2.70) (Mann and Picard 1994, Szeliski 1996). We also saw how a camera undergoing a pure rotation induces a different kind of homography (2.71).

In this section, I review both of these models and show how they can be applied to different stitching situations. I also introduce spherical and cylindrical compositing surfaces and show how, under favorable circumstances, these can be used to perform alignment using pure translations (§8.1.5).

8.1.1 Planar perspective motion

The simplest possible motion model to use with images is to simply translate and rotate them in 2D (Figure 8.2a). This is exactly the same kind of motion that you would use if you had overlapping

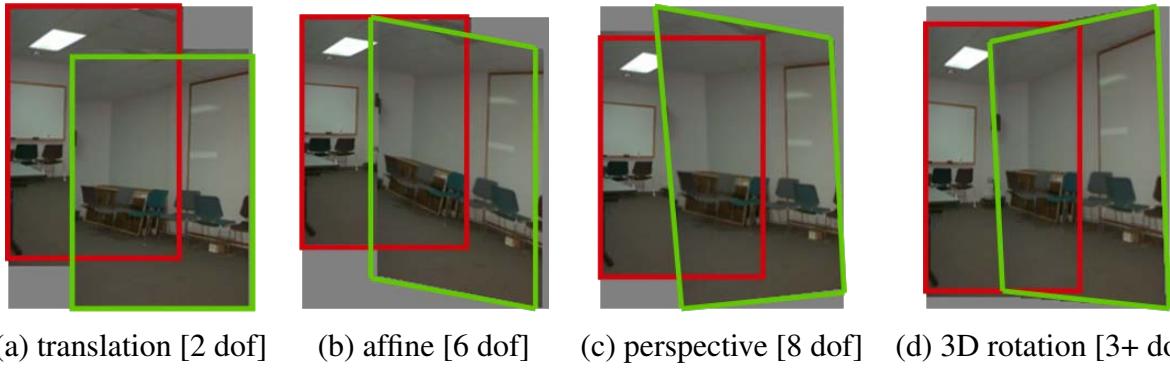


Figure 8.2: Two-dimensional motion models and how they can be used for image stitching.

photographic prints. It is also the kind of technique favored by David Hockney to create the collages that he calls *joiners* (Zelnik-Manor and Perona 2007, Nomura *et al.* 2007). Creating such collages, which show visible seams and inconsistencies that add to the artistic effect, is popular on Web sites such as Flickr, where they more commonly go under the name *panography* §5.1.2. Translation and rotation are also usually adequate motion models to compensate for small camera motions in applications such as photo and video stabilization and merging (§5.1.4 and §7.2.1).

In §5.1.3, we saw how the mapping between two cameras viewing a common plane can be described using a 3×3 homography (2.70). In particular, this matrix arises when mapping a pixel in one image to a 3D point and then back onto a second image,

$$\tilde{x}_1 \sim \tilde{P}_1 \tilde{P}_0^{-1} \tilde{x}_0 = M_{10} \tilde{x}_0. \quad (8.1)$$

When the last row of the P_0 matrix is replaced with a plane equation $\hat{n}_0 \cdot p + c_0$ and points are assumed to lie on this plane, i.e., their disparity is $d_0 = 0$, we can ignore the last column of M_{10} and also its last row, since we do not care about the final z-buffer depth. The resulting homography matrix \tilde{H}_{10} (the upper left 3×3 sub-matrix of M_{10}) describes the mapping between pixels in the two images,

$$\tilde{x}_1 \sim \tilde{H}_{10} \tilde{x}_0. \quad (8.2)$$

This observation formed the basis of some of the earliest automated image stitching algorithms (Mann and Picard 1994, Szeliski 1994, Szeliski 1996). Because reliable feature matching techniques had not yet been developed, these algorithms used direct pixel value matching, i.e., parametric motion estimation, as described in §7.2 and (7.53–7.54), to perform the image alignment.

More recent stitching algorithms first extract features and then match them up, often using robust techniques such as RANSAC (§5.1.5) to compute a good set of inliers. The final computation of the homography (8.2), i.e., the solution of the least squares fitting problem given pairs of

Note: Example of document scanning stitching

Figure 8.3: *An example of stitching together four pieces of a larger artwork obtained from a flatbed scanner.*

[Note: Generate this, using VideoMosaic so you can see the outlines]

corresponding features,

$$x_1 = \frac{(1 + h_{00})x_0 + h_{01}y_0 + h_{02}}{h_{20}x_0 + h_{21}y_0 + 1} \quad \text{and} \quad y_1 = \frac{h_{10}x_0 + (1 + h_{11})y_0 + h_{12}}{h_{20}x_0 + h_{21}y_0 + 1}, \quad (8.3)$$

uses iterative least squares, as described in §5.1.3 and (5.6–5.9).

8.1.2 Application: Whiteboard and document scanning

The simplest kind of image stitching to perform is to stitch together a number of image scans taken on a flatbed scanner. Say you have a large map, or a piece of child's artwork, that is too large to fit on your scanner. Simply take multiple scans of the document, making sure to overlap the scans by a large enough amount to ensure that the feature matching will work. Next, take successive pairs of images that you know overlap, extract features, match them up, and estimate the 2D rigid transform (2.16),

$$\mathbf{x}_{k+1} = \mathbf{R}_k \mathbf{x}_k + \mathbf{t}_k, \quad (8.4)$$

that best matches the features, using two-point RANSAC, if necessary, to find a good set of inliers. Then, on a final compositing surface (say aligned with the first scan), resample your images (§3.5.1) and average them together. Figure 8.3 shows an example of a composite created using this process.

Can you see any potential problems with this scheme?

One complication is that a 2D rigid transformation is non-linear in the rotation angle θ , so you will have to either use non-linear least squares or constraint \mathbf{R} to be orthonormal, as described in §5.1.3.

A bigger problem lies in the pairwise alignment process. As you align more and more pairs, the solution may drift so that it is no longer globally consistent. In this case, a global optimization procedure, as described in §8.2, may be required. Such global optimization often requires a large system of non-linear equations to be solved, although in some cases, such a linearized homographies (below) or similarity transforms (§5.1.2), regular least-squares may be an option.

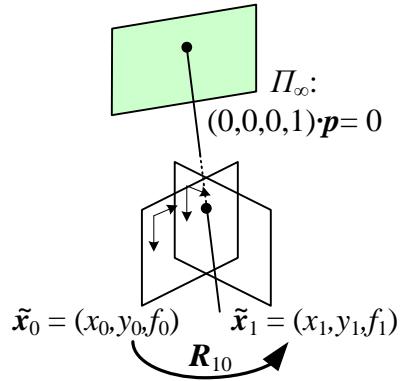


Figure 8.4: *Pure 3D camera rotation. The form of the homography (mapping) is particularly simple and depends only on the 3D rotation matrix and focal lengths.*

A slightly more complex scenario is when you take multiple overlapping handheld pictures of a whiteboard or other large planar object (He and Zhang 2005, Zhang and He 2007). Here, the natural motion model to use is a homography, although a more complex model that estimates the 3D rigid motion relative to the plane (plus the focal length, if unknown), could in principle be used.

8.1.3 Rotational panoramas

The most typical case for panoramic image stitching is when the camera undergoes a pure rotation. Think of standing at the rim of the Grand Canyon. Relative to the distant geometry in the scene, the camera is undergoing a pure rotation as you snap away. which is equivalent to assuming all points are very far from the camera, i.e., on the *plane at infinity* (Figure 8.4). Setting $t_0 = t_1 = 0$, we get the simplified 3×3 homography

$$\tilde{\mathbf{H}}_{10} = \mathbf{K}_1 \mathbf{R}_1 \mathbf{R}_0^{-1} \mathbf{K}_0^{-1} = \mathbf{K}_1 \mathbf{R}_{10} \mathbf{K}_0^{-1}, \quad (8.5)$$

where $\mathbf{K}_k = \text{diag}(f_k, f_k, 1)$ is the simplified camera intrinsic matrix (Szeliski 1996). This can also be re-written as

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_1 & & \\ & f_1 & \\ & & 1 \end{bmatrix} \mathbf{R}_{10} \begin{bmatrix} f_0^{-1} & & \\ & f_0^{-1} & \\ & & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \quad (8.6)$$

or

$$\begin{bmatrix} x_1 \\ y_1 \\ f_1 \end{bmatrix} \sim \mathbf{R}_{10} \begin{bmatrix} x_0 \\ y_0 \\ f_0 \end{bmatrix}, \quad (8.7)$$

which reveals the simplicity of the mapping equations and makes all of the motion parameters explicit. Thus, instead of the general 8-parameter homography relating a pair of images, we get the

3-, 4-, or 5-parameter *3D rotation* motion models corresponding to the cases where the focal length f is known, fixed, or variable (Szeliski and Shum 1997). Estimating the 3D rotation matrix (and optionally, focal length) associated with each image is intrinsically more stable than estimating a full 8-d.o.f. homography, which makes this the method of choice for large-scale image stitching algorithms (Szeliski and Shum 1997, Shum and Szeliski 2000, Brown and Lowe 2003a).

Given this representation, how do we update the rotation matrices to best align two overlapping images?

Recall from (2.70–8.5) that the equations relating two views can be written as

$$\tilde{\mathbf{x}}_1 \sim \tilde{\mathbf{H}}_{10} \tilde{\mathbf{x}}_0 \text{ with } \tilde{\mathbf{H}}_{10} = \mathbf{K}_1 \mathbf{R}_{10} \mathbf{K}_0^{-1}, \quad (8.8)$$

where $\mathbf{K}_k = \text{diag}(f_k, f_k, 1)$ is the calibration matrix and $\mathbf{R}_{10} = \mathbf{R}_1 \mathbf{R}_0^{-1}$ is rotation *between* the two views. The best way to update \mathbf{R}_{10} is to prepend an *incremental* rotation matrix $\mathbf{R}(\omega)$ to the current estimate \mathbf{R}_{10} (Szeliski and Shum 1997, Shum and Szeliski 2000),

$$\tilde{\mathbf{H}}(\omega) = \mathbf{K}_1 \mathbf{R}(\omega) \mathbf{R}_{10} \mathbf{K}_0^{-1} = [\mathbf{K}_1 \mathbf{R}(\omega) \mathbf{K}_1^{-1}] [\mathbf{K}_1 \mathbf{R}_{10} \mathbf{K}_0^{-1}] = \mathbf{D} \tilde{\mathbf{H}}_{10}. \quad (8.9)$$

Note that here I have written the update rule in the *compositional* form, where the incremental update \mathbf{D} is *prepended* to the current homography $\tilde{\mathbf{H}}_{10}$. Using the small-angle approximation to $\mathbf{R}(\omega)$ given in (2.35), we can write the incremental update matrix as

$$\mathbf{D} = \mathbf{K}_1 \mathbf{R}(\omega) \mathbf{K}_1^{-1} \approx \mathbf{K}_1 (\mathbf{I} + [\omega]_\times) \mathbf{K}_1^{-1} = \begin{bmatrix} 1 & -\omega_z & f_1 \omega_y \\ \omega_z & 1 & -f_1 \omega_x \\ -\omega_y/f_1 & \omega_x/f_1 & 1 \end{bmatrix}. \quad (8.10)$$

Notice how there is now a nice one-to-one correspondence between the entries in the \mathbf{D} matrix and the h_{00}, \dots, h_{21} parameters used in Table 7.1 and (5.5), i.e.,

$$(h_{00}, h_{01}, h_{02}, h_{00}, h_{11}, h_{12}, h_{20}, h_{21}) = (0, -\omega_z, f_1 \omega_y, \omega_z, 0, -f_1 \omega_x, -\omega_y/f_1, \omega_x/f_1). \quad (8.11)$$

We can therefore apply the chain rule to (5.10) and (8.11) to obtain

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} -xy/f_1 & f_1 + x^2/f_1 & -y \\ -(f_1 + y^2/f_1) & xy/f_1 & x \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad (8.12)$$

which give us the linearized update equations needed to estimate $\omega = (\omega_x, \omega_y, \omega_z)$.¹ Notice that this update rule depends on the focal length f_1 of the *target* view, and is independent of the focal

¹ This is the same as the rotational component of instantaneous rigid flow (Bergen *et al.* 1992) and the same as the update equations given in (Szeliski and Shum 1997, Shum and Szeliski 2000).

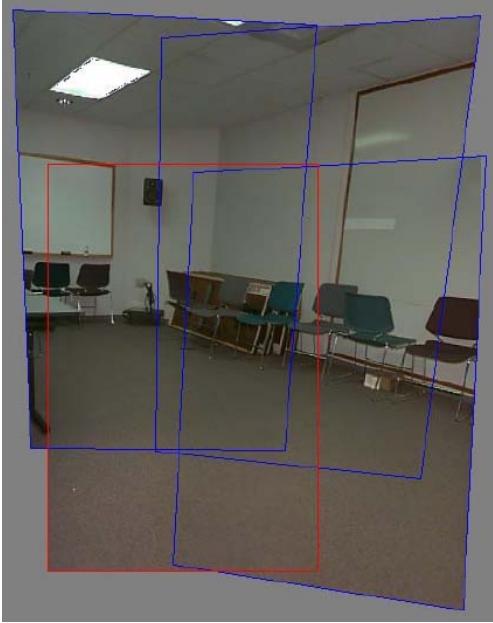


Figure 8.5: *Four images taken with a hand-held model registered using a 3D rotation motion model (from (Szeliski and Shum 1997)). Notice how the homographies, rather than being arbitrary, have a well defined keystone shape whose width increases away from the origin.*

length f_0 of the *template* view. This is because the compositional algorithm essentially makes small perturbations to the target. Once the incremental rotation vector ω has been computed, the \mathbf{R}_1 rotation matrix can be updated using $\mathbf{R}_1 \leftarrow \mathbf{R}(\omega)\mathbf{R}_1$.

The formulas for updating the focal length estimates are a little more involved and are given in (Shum and Szeliski 2000). I will not repeat them here, since an alternative update rule, based on minimizing the difference between back-projected 3D rays, will be given in §8.2.1. Figure 8.5 shows the alignment of four images under the 3D rotation motion model.

Focal length initialization. In order to initialize the 3D rotation model, we need to simultaneously estimate the focal length(s) of the camera(s) and an initial guess for a rotation matrix. This can be obtained directly from a homography-based (planar perspective) alignment $\tilde{\mathbf{H}}_{10}$, using the formulas first presented in (Szeliski and Shum 1997).

Using the simplified form of the calibration matrices $\mathbf{K}_k = \text{diag}(f_k, f_k, 1)$ first used in (8.5), we can rewrite (8.8) as

$$\mathbf{R}_{10} \sim \mathbf{K}_1^{-1} \tilde{\mathbf{H}}_{10} \mathbf{K}_0 \sim \begin{bmatrix} h_{00} & h_{01} & f_0^{-1} h_{02} \\ h_{10} & h_{11} & f_0^{-1} h_{12} \\ f_1 h_{20} & f_1 h_{21} & f_0^{-1} f_1 h_{22} \end{bmatrix}, \quad (8.13)$$

where the h_{ij} are the elements of $\tilde{\mathbf{H}}_{10}$.

Using the orthonormality properties of the rotation matrix \mathbf{R}_{10} and the fact that the right hand side of (8.13) is known only up to a scale, we obtain

$$h_{00}^2 + h_{01}^2 + f_0^{-2}h_{02}^2 = h_{10}^2 + h_{11}^2 + f_0^{-2}h_{12}^2 \quad (8.14)$$

and

$$h_{00}h_{10} + h_{01}h_{11} + f_0^{-2}h_{02}h_{12} = 0. \quad (8.15)$$

From this, we can compute estimates for f_0 of

$$f_0^2 = \frac{h_{12}^2 - h_{02}^2}{h_{00}^2 + h_{01}^2 - h_{10}^2 - h_{11}^2} \text{ if } h_{00}^2 + h_{01}^2 \neq h_{10}^2 + h_{11}^2 \quad (8.16)$$

or

$$f_0^2 = -\frac{h_{02}h_{12}}{h_{00}h_{10} + h_{01}h_{11}} \text{ if } h_{00}h_{10} \neq -h_{01}h_{11}. \quad (8.17)$$

(Note that the equations given in (Szeliski and Shum 1997) are erroneous; the correct equations can be found in (Shum and Szeliski 2000).) If neither of these conditions holds, we can also take the dot products between the first (or second) row and the third one. Similar result can be obtained for f_1 as well by analyzing the columns of $\tilde{\mathbf{H}}_{10}$. If the focal length is the same for both images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1f_0}$. When multiple estimates of f are available, e.g., from different homographies, the median value can be used as the final estimate.

Gap closing. The techniques presented in this section can be used to estimate a series of rotation matrices and focal lengths, which can be chained together to create large panoramas. Unfortunately, because of accumulated errors, this approach will rarely produce a closed 360° panorama. Instead, there will invariably be either a gap or an overlap (Figure 8.6).

We can solve this problem by matching the first image in the sequence with the last one. The difference between the two rotation matrix estimates associated with this frame indicates the amount of misregistration. This error can be distributed evenly across the whole sequence by taking the quotient of the two quaternions associated with these rotations and dividing this “error quaternion” by the number of images in the sequence (assuming relatively constant inter-frame rotations). We can also update the estimated focal length based on the amount of misregistration. To do this, we first convert the error quaternion into a *gap angle*, θ_g . We then update the focal length using the equation $f' = f(1 - \theta_g/360^\circ)$.

Figure 8.6a shows the end of registered image sequence and the first image. There is a big gap between the last image and the first which are in fact the same image. The gap is 32° because the wrong estimate of focal length ($f = 510$) was used. Figure 8.6b shows the registration after

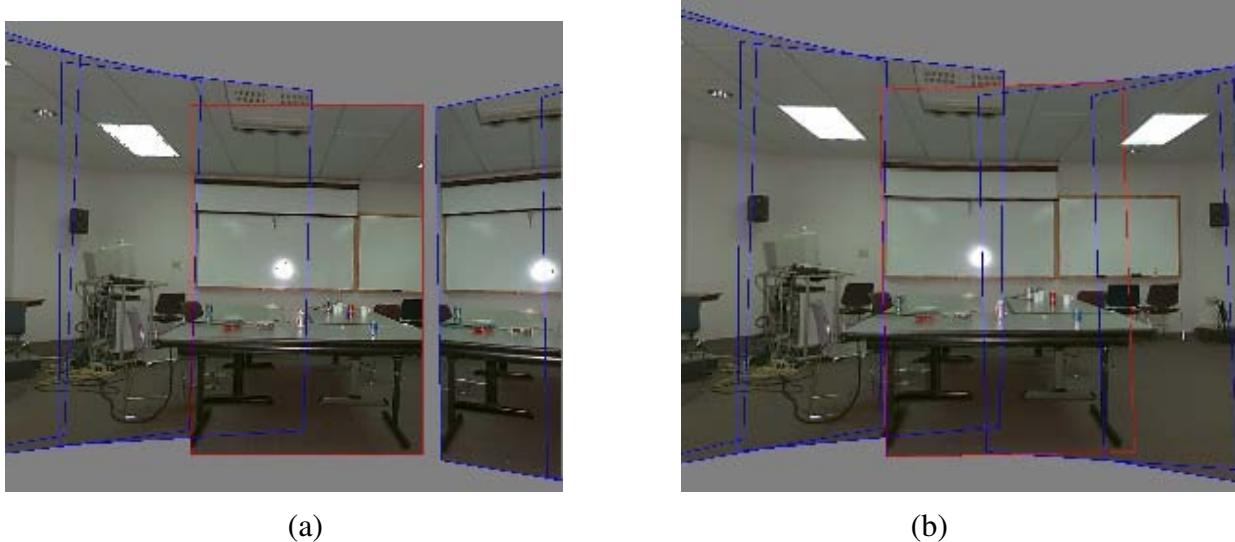


Figure 8.6: *Gap closing*: (a) a gap is visible when the focal length is wrong ($f = 510$); (b) no gap is visible for the correct focal length ($f = 468$).

closing the gap with the correct focal length ($f = 468$). Notice that both mosaics show very little visual misregistration (except at the gap), yet Figure 8.6a has been computed using a focal length which has 9% error. Related approaches have been developed by (Hartley 1994c, McMillan and Bishop 1995, Stein 1995, Kang and Weiss 1997) to solve the focal length estimation problem using pure panning motion and cylindrical images.

Unfortunately, this particular gap-closing heuristic only works for the kind of “one-dimensional” panorama where the camera is continuously turning in the same direction. In next section §8.2, I describe a different approach to removing gaps and overlaps that works for arbitrary camera motions.

8.1.4 Application: Video summarization and compression

An interesting application of image stitching is the ability to summarize and compress videos taken with a panning camera. This application was first suggested by Teodosio and Bender (1993), who called their mosaic-based summaries *salient stills*. These ideas were then extended by Irani *et al.* (Irani *et al.* 1995, Kumar *et al.* 1995, Irani and Anandan 1998) to additional applications such as video compression and video indexing. While these early approaches used affine motion models and were therefore restricted to long focal lengths, the techniques were generalized by Lee *et al.* (1997) to full 8-parameter homographies and incorporated into the MPEG-4 video compression standard, where the stitched background layers were called *video sprites* (Figure 8.7).

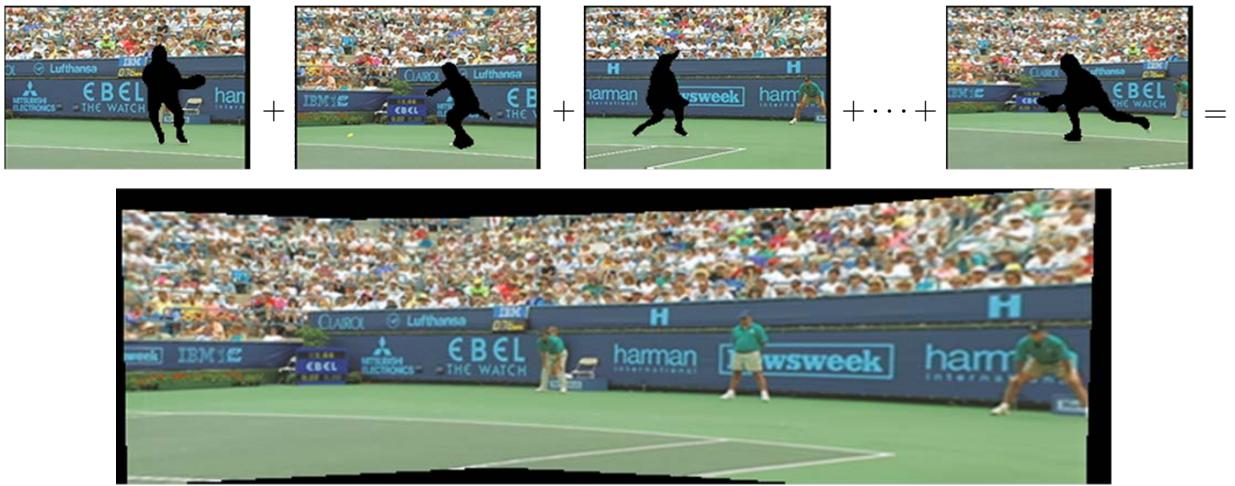


Figure 8.7: *Video stitching the background scene to create a single sprite image that can be transmitted and used to re-create the background in each frame (Lee et al. 1997).*

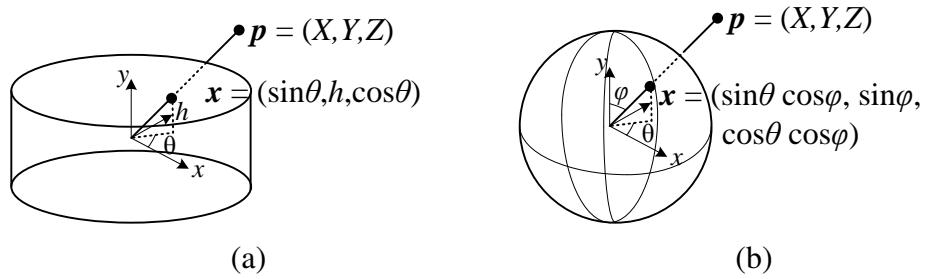


Figure 8.8: *Projection from 3D to cylindrical and spherical coordinates.*

8.1.5 Cylindrical and spherical coordinates

An alternative to using homographies or 3D motions to align images is to first warp the images into *cylindrical* coordinates and to then use a pure translational model to align them (Chen 1995, Szeliski 1996). Unfortunately, this only works if the images are all taken with a level camera or with a known tilt angle.

Assume for now that the camera is in its canonical position, i.e., its rotation matrix is the identity, $R = I$, so that the optical axis is aligned with the z axis and the y axis is aligned vertically. The 3D ray corresponding to an (x, y) pixel is therefore (x, y, f) .

We wish to project this image onto a *cylindrical surface* of unit radius (Szeliski 1996). Points on this surface are parameterized by an angle θ and a height h , with the 3D cylindrical coordinates corresponding to (θ, h) given by

$$(\sin \theta, h, \cos \theta) \propto (x, y, f), \quad (8.18)$$

as shown in Figure 8.8a. From this correspondence, we can compute the formula for the *warped* or *mapped* coordinates (Szeliski and Shum 1997),

$$x' = s\theta = s \tan^{-1} \frac{x}{f}, \quad (8.19)$$

$$y' = sh = s \frac{y}{\sqrt{x^2 + f^2}}, \quad (8.20)$$

where s is an arbitrary scaling factor (sometimes called the *radius* of the cylinder) that can be set to $s = f$ to minimize the distortion (scaling) near the center of the image.² The inverse of this mapping equation is given by

$$x = f \tan \theta = f \tan \frac{x'}{s}, \quad (8.21)$$

$$y = h \sqrt{x^2 + f^2} = \frac{y'}{s} f \sqrt{1 + \tan^2 x'/s} = f \frac{y'}{s} \sec \frac{x'}{s}. \quad (8.22)$$

Images can also be projected onto a *spherical surface* (Szeliski and Shum 1997), which is useful if the final panorama includes a full sphere or hemisphere of views, instead of just a cylindrical strip. In this case, the sphere is parameterized by two angles (θ, ϕ) , with 3D spherical coordinates given by

$$(\sin \theta \cos \phi, \sin \phi, \cos \theta \cos \phi) \propto (x, y, f), \quad (8.23)$$

as shown in Figure 8.8b.³ The correspondence between coordinates is now given by (Szeliski and Shum 1997)

$$x' = s\theta = s \tan^{-1} \frac{x}{f}, \quad (8.24)$$

$$y' = s\phi = s \tan^{-1} \frac{y}{\sqrt{x^2 + f^2}}, \quad (8.25)$$

while the inverse is given by

$$x = f \tan \theta = f \tan \frac{x'}{s}, \quad (8.26)$$

$$y = \sqrt{x^2 + f^2} \tan \phi = \tan \frac{y'}{s} f \sqrt{1 + \tan^2 x'/s} = f \tan \frac{y'}{s} \sec \frac{x'}{s}. \quad (8.27)$$

Note that it may be simpler to generate a scaled (x, y, z) direction from (8.23) followed by a perspective division by z and a scaling by f .

² The scale can also be set to a larger or smaller value for the final compositing surface, depending on the desired output panorama resolution—see §8.3.

³ Note that these are not the usual spherical coordinates first presented in (2.8). Here, the y axis points at the north pole instead of the z axis, since we are used to viewing images taken horizontally, i.e., with the y axis pointing in the direction of the gravity vector.

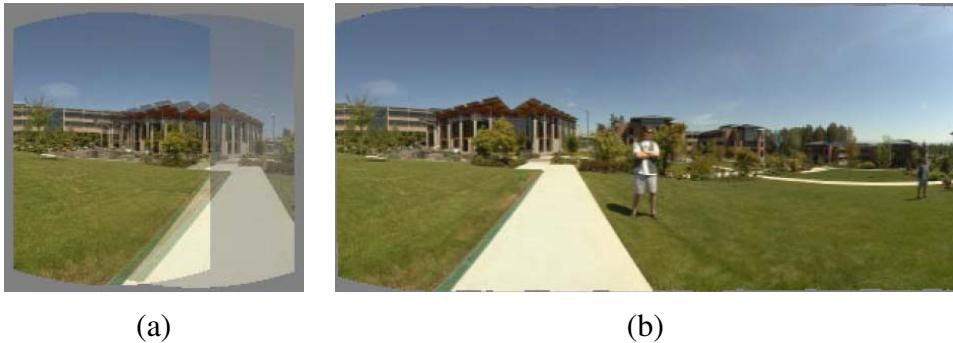


Figure 8.9: *An example of a cylindrical panorama: (a) two cylindrically warped images related by a horizontal translation; (b) part of a cylindrical panorama composited from a sequence of images.*

Cylindrical image stitching algorithms are most commonly used when the camera is known to be level and only rotating around its vertical axis (Chen 1995). Under these conditions, images at different rotations are related by a pure horizontal translation.⁴ This makes it attractive as an initial class project in an introductory computer vision course, since the full complexity of the perspective alignment algorithm (§7.2 & §5.1) can be avoided. Figure 8.9 shows how two cylindrically warped images from a leveled rotational panorama are related by a pure translation (Szeliski and Shum 1997).

Professional panoramic photographers sometimes also use a pan-tilt head that makes it easy to control the tilt and to stop at specific *detents* in the rotation angle. This not only ensures a uniform coverage of the visual field with a desired amount of image overlap, but also makes it possible to stitch the images using cylindrical or spherical coordinates and pure translations. In this case, pixel coordinates (x, y, f) must first be rotated using the known tilt and panning angles before being projected into cylindrical or spherical coordinates (Chen 1995). Having a roughly known panning angle also makes it easier to compute the alignment, since the rough relative positioning of all the input images is known ahead of time, enabling a reduced search range for alignment. Figure 8.10 shows a full 3D rotational panorama unwrapped onto the surface of a sphere (Szeliski and Shum 1997).

One final coordinate mapping worth mentioning is the *polar* mapping where the north pole lies along the optical axis rather than the vertical axis,

$$(\cos \theta \sin \phi, \sin \theta \sin \phi, \cos \phi) = s(x, y, z). \quad (8.28)$$

In this case, the mapping equations become

$$x' = s\phi \cos \theta = s \frac{x}{r} \tan^{-1} \frac{r}{z}, \quad (8.29)$$

⁴Small vertical tilts can sometimes be compensated for with vertical translations.



Figure 8.10: An example of a spherical panorama constructed from 54 photographs.

Figure 8.11: An example of a 3D rotational panorama mapped using a polar mapping.
 [Note: Need to write the code to generate this, or just use HDView!]

$$y' = s\phi \sin \theta = s \frac{y}{r} \tan^{-1} \frac{r}{z}, \quad (8.30)$$

where $r = \sqrt{x^2 + y^2}$ is the *radial distance* in the (x, y) plane and $s\phi$ plays a similar role in the (x', y') plane. This mapping provides an attractive visualization surface for certain kinds of wide-angle panoramas and is also a good model for the distortion induced by *fisheye lenses*, as discussed in §2.1.5. Note how for small values of (x, y) , the mapping equations reduces to $x' \approx sx/z$, which suggests that s plays a role similar to the focal length f . [Note: Figure 8.11 shows the full 3D rotational panorama shown in Figure 8.10 unwrapped onto a polar compositing surface.]

Cylindrical strip panoramas

To generate cylindrical or spherical panoramas from a horizontally panning (rotating) camera, it's best to use a tripod. Set your camera up to take a series of 50% overlapped photos, and then use the following steps to create your panorama

1. Estimate the amount of radial distortion by taking some pictures with lots of long straight lines near the edges of the image. Use (2.75) in §2.1.5 to undistort the image, and adjust your κ_1 (and optionally κ_2)
2. Compute the focal length either by finding

[Note: Guess the focal length (use a ruler trick).

Level your tripod.

Warp to cylindrical.

Align using a feature-based, or FFT technique.

Blend with averaging (can use better blend modes from §8.3.3 later).

Or, should this be an exercise?]

8.2 Global alignment

So far, I have discussed how to register pairs of images using both direct and feature-based methods using a variety of motion models. In most applications, we are given more than a single pair of images to register. The goal is then to find a *globally consistent* set of alignment parameters that minimize the mis-registration between all pairs of images (Szeliski and Shum 1997, Shum and Szeliski 2000, Sawhney and Kumar 1999, Coorg and Teller 2000). In order to do this, we need to extend the pairwise matching criteria (7.1), (7.50), and (5.1) to a global energy function that involves all of the per-image pose parameters (§8.2.1). Once we have computed the global alignment, we often need to perform *local adjustments* such as *parallax removal* to reduce double images and blurring due to local mis-registrations (§8.2.2). Finally, if we are given an unordered set of images to register, we need to discover which images go together to form one or more panoramas. This process of *panorama recognition* is described in §8.2.3.

8.2.1 Bundle adjustment

One way to register a large number of images is to add new images to the panorama one at a time, aligning the most recent image with the previous ones already in the collection (Szeliski and Shum 1997), and discovering, if necessary, which images it overlaps (Sawhney and Kumar 1999). In the case of 360° panoramas, accumulated error may lead to the presence of a *gap* (or excessive overlap) between the two ends of the panorama, which can be fixed by stretching the alignment of all the images using a process called *gap closing* (Szeliski and Shum 1997). However, a better alternative is to simultaneously align all the images together using a least squares framework to correctly distribute any mis-registration errors.

The process of simultaneously adjusting pose parameters for a large collection of overlapping images is called *bundle adjustment* in the photogrammetry community (Triggs *et al.* 1999). In computer vision, it was first applied to the general structure from motion problem (Szeliski and Kang 1994) and then later specialized for panoramic image stitching (Shum and Szeliski 2000, Sawhney and Kumar 1999, Coorg and Teller 2000).

In this section, I formulate the problem of global alignment using a feature-based approach, since this results in a simpler system. An equivalent direct approach can be obtained either by dividing images into patches and creating a virtual feature correspondence for each one (as discussed in §8.2.5 and (Shum and Szeliski 2000)), or by replacing the per-feature error metrics with per-pixel metrics.

Consider the feature-based alignment problem given in (5.1), i.e.,

$$E_{\text{pairwise-LS}} = \sum_i \|\mathbf{r}_i\|^2 = \|\tilde{\mathbf{x}}'_i(\mathbf{x}_i; \mathbf{p}) - \hat{\mathbf{x}}'_i\|^2. \quad (8.31)$$

For multi-image alignment, instead of having a single collection of pairwise feature correspondences, $\{(\mathbf{x}_i, \hat{\mathbf{x}}'_i)\}$, we have a collection of n features, with the location of the i th feature point in the j th image denoted by \mathbf{x}_{ij} and its scalar confidence (inverse variance) denoted by c_{ij} .⁵ Each image also has some associated *pose* parameters.

In this section, I assume that this pose consists of a rotation matrix \mathbf{R}_j and a focal length f_j , although formulations in terms of homographies are also possible (Shum and Szeliski 1997, Sawhney and Kumar 1999). The equation mapping a 3D point \mathbf{x}_i into a point \mathbf{x}_{ij} in frame j can be re-written from (2.67–8.5) as

$$\tilde{\mathbf{x}}_{ij} \sim \mathbf{K}_j \mathbf{R}_j \mathbf{x}_i \quad \text{and} \quad \mathbf{x}_i \sim \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_{ij}, \quad (8.32)$$

where $\mathbf{K}_j = \text{diag}(f_j, f_j, 1)$ is the simplified form of the calibration matrix. The motion mapping a point \mathbf{x}_{ij} from frame j into a point \mathbf{x}_{ik} in frame k is similarly given by

$$\tilde{\mathbf{x}}_{ik} \sim \tilde{\mathbf{H}}_{kj} \tilde{\mathbf{x}}_{ij} = \mathbf{K}_k \mathbf{R}_k \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} \tilde{\mathbf{x}}_{ij}. \quad (8.33)$$

Given an initial set of $\{(\mathbf{R}_j, f_j)\}$ estimates obtained from chaining pairwise alignments, how do we refine these estimates?

One approach is to directly extend the pairwise energy $E_{\text{pairwise-LS}}$ (8.31) to a multiview formulation,

$$E_{\text{all-pairs-2D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\mathbf{x}}_{ik}(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j, \mathbf{R}_k, f_k) - \hat{\mathbf{x}}_{ik}\|^2, \quad (8.34)$$

where the $\tilde{\mathbf{x}}_{ik}$ function is the *predicted* location of feature i in frame k given by (8.33), $\hat{\mathbf{x}}_{ij}$ is the *observed* location, and the “2D” in the subscript indicates than an image-plane error is being minimized (Shum and Szeliski 1997). [Note: I’m overloading the use of tilde and hat here, sometimes for predicted/observed, and sometimes for homogeneous/normalized. I’m not sure how to get around this.] Note that since $\tilde{\mathbf{x}}_{ik}$ depends on the $\hat{\mathbf{x}}_{ij}$ observed value, we actually have an *errors-in-variable* problem, which in principle requires more sophisticated techniques than least

⁵ Features that not seen in image j have $c_{ij} = 0$. We can also use 2×2 inverse covariance matrices Σ_{ij}^{-1} in place of c_{ij} , as shown in (5.3).

squares to solve. [Note: Add citation to Meer, errors-in-variables.] However, in practice, if we have enough features, we can directly minimize the above quantity using regular non-linear least squares and obtain an accurate multi-frame alignment.⁶

While this approach works well in practice, it suffers from two potential disadvantages. First, since a summation is taken over all pairs with corresponding features, features that are observed many times get overweighted in the final solution. (In effect, a feature observed m times gets counted $\binom{m}{2}$ times instead of m times.) Second, the derivatives of $\tilde{\mathbf{x}}_{ik}$ w.r.t. the $\{(\mathbf{R}_j, f_j)\}$ are a little cumbersome, although using the incremental correction to \mathbf{R}_j introduced in §8.1.3 makes this more tractable.

An alternative way to formulate the optimization is to use true bundle adjustment, i.e., to solve not only for the pose parameters $\{(\mathbf{R}_j, f_j)\}$ but also for the 3D point positions $\{\mathbf{x}_i\}$,

$$E_{\text{BA-2D}} = \sum_i \sum_j c_{ij} \|\tilde{\mathbf{x}}_{ij}(\mathbf{x}_i; \mathbf{R}_j, f_j) - \hat{\mathbf{x}}_{ij}\|^2, \quad (8.35)$$

where $\tilde{\mathbf{x}}_{ij}(\mathbf{x}_i; \mathbf{R}_j, f_j)$ is given by (8.32). The disadvantage of full bundle adjustment is that there are more variables to solve for, so both each iteration and the overall convergence may be slower. (Imagine how the 3D points need to “shift” each time some rotation matrices are updated.) However, the computational complexity of each linearized Gauss-Newton step can be reduced using sparse matrix techniques (Szeliski and Kang 1994, Hartley and Zisserman 2004, Triggs *et al.* 1999).

An alternative formulation is to minimize the error in 3D projected ray directions (Shum and Szeliski 2000), i.e.,

$$E_{\text{BA-3D}} = \sum_i \sum_j c_{ij} \|\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) - \mathbf{x}_i\|^2, \quad (8.36)$$

where $\tilde{\mathbf{x}}_i(\mathbf{x}_{ij}; \mathbf{R}_j, f_j)$ is given by the second half of (8.32). This in itself has no particular advantage over (8.35). In fact, since errors are being minimized in 3D ray space, there is a bias towards estimating longer focal lengths, since the angles between rays become smaller as f increases.

However, if we eliminate the 3D rays \mathbf{x}_i , we can derive a pairwise energy formulated in 3D ray space (Shum and Szeliski 2000),

$$E_{\text{all-pairs-3D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \|\tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j) - \tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ik}; \mathbf{R}_k, f_k)\|^2. \quad (8.37)$$

This results in the simplest set of update equations (Shum and Szeliski 2000), since the f_k can be folded into the creation of the homogeneous coordinate vector as in (8.7). Thus, even though this

⁶ While there exists an overall pose ambiguity in the solution, i.e., all the \mathbf{R}_j can be post-multiplied by an arbitrary rotation \mathbf{R}_g , a well-conditioned non-linear least squares algorithm such as Levenberg Marquardt will handle this degeneracy without trouble.

formula over-weights features that occur more frequently, it is the method used both by [Shum and Szeliski \(2000\)](#) and in our current work ([Brown et al. 2005](#)). In order to reduce the bias towards longer focal lengths, I multiply each residual (3D error) by $\sqrt{f_j f_k}$, which is similar to projecting the 3D rays into a “virtual camera” of intermediate focal length, and which seems to work well in practice. [Note: Give more details here, as in the class project assignment?] [Note: Skip the up vector estimation (below) in the book.]

Up vector selection. As mentioned above, there exists a global ambiguity in the pose of the 3D cameras computed by the above methods. While this may not appear to matter, people have a preference for the final stitched image being “upright” rather than twisted or tilted. More concretely, people are used to seeing photographs displayed so that the vertical (gravity) axis points straight up in the image. Consider how you usually shoot photographs: while you may pan and tilt the camera any which way, you usually keep vertical scene lines parallel to the vertical edge of the image. In other words, the horizontal edge of your camera (its x -axis) usually stays parallel to the ground plane (perpendicular to the world gravity direction).

Mathematically, this constraint on the rotation matrices can be expressed as follows. Recall from (8.32) that the 3D→2D projection is given by

$$\tilde{\mathbf{x}}_{ik} \sim \mathbf{K}_k \mathbf{R}_k \mathbf{x}_i. \quad (8.38)$$

We wish to post-multiply each rotation matrix \mathbf{R}_k by a global rotation \mathbf{R}_g such that the projection of the global y -axis, $\hat{\mathbf{j}} = (0, 1, 0)$ is perpendicular to the image x -axis, $\hat{\mathbf{i}} = (1, 0, 0)$.⁷

This constraint can be written as

$$\hat{\mathbf{i}}^T \mathbf{R}_k \mathbf{R}_g \hat{\mathbf{j}} = 0 \quad (8.39)$$

(note that the scaling by the calibration matrix is irrelevant here). This is equivalent to requiring that the first row of \mathbf{R}_k , $\mathbf{r}_{k0} = \hat{\mathbf{i}}^T \mathbf{R}_k$ be perpendicular to the second column of \mathbf{R}_g , $\mathbf{r}_{g1} = \mathbf{R}_g \hat{\mathbf{j}}$. This set of constraints (one per input image) can be written as a least squares problem,

$$\mathbf{r}_{g1} = \arg \min_{\mathbf{r}} \sum_k (\mathbf{r}^T \mathbf{r}_{k0})^2 = \arg \min_{\mathbf{r}} \mathbf{r}^T \left[\sum_k \mathbf{r}_{k0} \mathbf{r}_{k0}^T \right] \mathbf{r}. \quad (8.40)$$

Thus, \mathbf{r}_{g1} is the smallest eigenvector of the *scatter* or *moment* matrix spanned by the individual camera rotation x -vectors, which should generally be of the form $(c, 0, s)$ when the cameras are upright.

To fully specify the \mathbf{R}_g global rotation, we need to specify one additional constraint. This is related to the *view selection* problem discussed in §8.3.1. One simple heuristic is to prefer the

⁷ Note that here we use the convention common in computer graphics that the vertical world axis corresponds to y . This is a natural choice if we wish the rotation matrix associated with a “regular” image taken horizontally to be the identity, rather than a 90° rotation around the x -axis.

average z -axis of the individual rotation matrices, $\bar{\mathbf{k}} = \sum_k \hat{\mathbf{k}}^T \mathbf{R}_k$ to be close to the world z -axis, $\mathbf{r}_{g2} = \mathbf{R}_g \hat{\mathbf{k}}$. We can therefore compute the full rotation matrix \mathbf{R}_g in three steps:

1. $\mathbf{r}_{g1} = \min \text{ eigenvector } (\sum_k \mathbf{r}_{k0} \mathbf{r}_{k0}^T);$
2. $\mathbf{r}_{g0} = \mathcal{N}((\sum_k \mathbf{r}_{k2}) \times \mathbf{r}_{g1});$
3. $\mathbf{r}_{g2} = \mathbf{r}_{g0} \times \mathbf{r}_{g1},$

where $\mathcal{N}(\mathbf{v}) = \mathbf{v}/\|\mathbf{v}\|$ normalizes a vector \mathbf{v} .

8.2.2 Parallax removal

Once we have optimized the global orientations and focal lengths of our cameras, we may find that the images are still not perfectly aligned, i.e., the resulting stitched image looks blurry or ghosted in some places. This can be caused by a variety of factors, including unmodeled radial distortion, 3D parallax (failure to rotate the camera around its optical center), small scene motions such as waving tree branches, and large-scale scene motions such as people moving in and out of pictures.

Each of these problems can be treated with a different approach. Radial distortion can be estimated (potentially before the camera's first use) using one of the techniques discussed in §2.1.5. For example, the *plumb line method* (Brown 1971, Kang 2001, El-Melegy and Farag 2003) adjusts radial distortion parameters until slightly curved lines become straight, while mosaic-based approaches adjust them until mis-registration is reduced in image overlap areas (Stein 1997, Sawhney and Kumar 1999).

3D parallax can be attacked by doing a full 3D bundle adjustment, i.e., replacing the projection equation (8.32) used in (8.35) with (2.67), which models camera translations. The 3D positions of the matched features points and cameras can then be simultaneously recovered, although this can be significantly more expensive than parallax-free image registration. Once the 3D structure has been recovered, the scene could (in theory) be projected to a single (central) viewpoint that contains no parallax. However, in order to do this, dense *stereo* correspondence needs to be performed (Kumar *et al.* 1995, Szeliski and Kang 1995, Scharstein and Szeliski 2002), which may not be possible if the images only contain partial overlap. In that case, it may be necessary to correct for parallax only in the overlap areas, which can be accomplished using a *Multi-Perspective Plane Sweep* (MPPS) algorithm (Kang *et al.* 2004, Uyttendaele *et al.* 2004).

When the motion in the scene is very large, i.e., when objects appear and disappear completely, a sensible solution is to simply *select* pixels from only one image at a time as the source for the final composite (Milgram 1977, Davis 1998, Agarwala *et al.* 2004), as discussed in §8.3.2. However, when the motion is reasonably small (on the order of a few pixels), general 2D motion estimation (optical flow) can be used to perform an appropriate correction before blending using a process

called *local alignment* (Shum and Szeliski 2000, Kang *et al.* 2003). This same process can also be used to compensate for radial distortion and 3D parallax, although it uses a weaker motion model than explicitly modeling the source of error, and may therefore fail more often or introduce unwanted distortions.

The local alignment technique introduced by Shum and Szeliski (2000) starts with the global bundle adjustment (8.37) used to optimize the camera poses. Once these have been estimated, the *desired* location of a 3D point \mathbf{x}_i can be estimated as the *average* of the back-projected 3D locations,

$$\bar{\mathbf{x}}_i \sim \sum_j c_{ij} \tilde{\mathbf{x}}_i(\hat{\mathbf{x}}_{ij}; \mathbf{R}_j, f_j), \quad (8.41)$$

which can be projected into each image j to obtain a *target location* $\bar{\mathbf{x}}_{ij}$. The difference between the target locations $\bar{\mathbf{x}}_{ij}$ and the original features \mathbf{x}_{ij} provide a set of local motion estimates

$$\mathbf{u}_{ij} = \bar{\mathbf{x}}_{ij} - \mathbf{x}_{ij}, \quad (8.42)$$

which can be interpolated to form a dense correction field $\mathbf{u}_j(\mathbf{x}_j)$. In their system, Shum and Szeliski (2000) use an *inverse warping* algorithm where the sparse $-\mathbf{u}_{ij}$ values are placed at the new target locations $\bar{\mathbf{x}}_{ij}$, interpolated using bilinear kernel functions (Nielson 1993) and then added to the original pixel coordinates when computing the warped (corrected) image. In order to get a reasonably dense set of features to interpolate, Shum and Szeliski (2000) place a feature point at the center of each patch (the patch size controls the smoothness in the local alignment stage), rather than relying of features extracted using an interest operator.

An alternative approach to motion-based de-ghosting was proposed by Kang *et al.* (2003), who estimate dense optical flow between each input image and a central *reference* image. The accuracy of the flow vector is checked using a photo-consistency measure before a given warped pixel is considered valid and therefore used to compute a high dynamic range radiance estimate, which is the goal of their overall algorithm. The requirement for having a reference image makes their approach less applicable to general image mosaicing, although an extension to this case could certainly be envisaged.

8.2.3 Recognizing panoramas

The final piece needed to perform fully automated image stitching is a technique to recognize which images actually go together, which Brown and Lowe (2003a) call *recognizing panoramas*. If the user takes images in sequence so that each image overlaps its predecessor and also specifies the first and last images to be stitched, bundle adjustment combined with the process of *topology inference* can be used to automatically assemble a panorama (Sawhney and Kumar 1999). However, users often jump around when taking panoramas, e.g., they may start a new row on top of

a previous one, or jump back to take a repeated shot, or create 360° panoramas where end-to-end overlaps need to be discovered. Furthermore, the ability to discover multiple panoramas taken by a user over an extended period of time can be a big convenience.

To recognize panoramas, Brown and Lowe (2003a) first find all pairwise image overlaps using a feature-based method and then find connected components in the overlap graph to “recognize” individual panoramas (Figure 8.12). The feature-based matching stage first extracts SIFT feature locations and feature descriptors (Lowe 2004) from all the input images and then places these in an indexing structure, as described in §4.1.3. For each image pair under consideration, the nearest matching neighbor is found for each feature in the first image, using the indexing structure to rapidly find candidates, and then comparing feature descriptors to find the best match. RANSAC is then used to find a set of *inlier* matches, using a pairs of matches to hypothesize a similarity motion model that is then used to count the number of inliers. (A more recent RANSAC algorithm tailored specifically for rotational panoramas is described in (Brown *et al.* 2007).) [Note: Need to re-read (Brown and Lowe 2003a) and ask Matt if this is right.]

In practice, the most difficult part of getting a fully automated stitching algorithm to work is deciding which pairs of images actually correspond to the same parts of the scene. Repeated structures such as windows (Figure 8.13) can lead to false matches when using a feature-based approach. One way to mitigate this problem is to perform a direct pixel-based comparison between the registered images to determine if they actually are different views of the same scene. Unfortunately, this heuristic may fail if there are moving objects in the scene (Figure 8.14). While there is no magic bullet for this problem short of full scene understanding, further improvements can likely be made by applying domain-specific heuristics such as priors on typical camera motions as well as machine learning techniques applied to the problem of match validation.

8.2.4 Application: Full-view panoramas and virtual environments

Cylindrical panoramas (QTVR, Surround Video, ...): special kind of environment map. IPix is two hemispheres (?)

Environment maps for CG backgrounds (video games?), reflection mapping (Szeliski and Shum 1997).

Also, Debevec’s “light probes” for CGI. Uses high dynamic range photography (§9.2), and more recently, single mirrored balls (faster)

8.2.5 Direct vs. feature-based alignment

Given that there exist these two alternative approaches to aligning images, which is preferable?

I used to be firmly in the direct matching camp (Irani and Anandan 1999). [Note: Need to

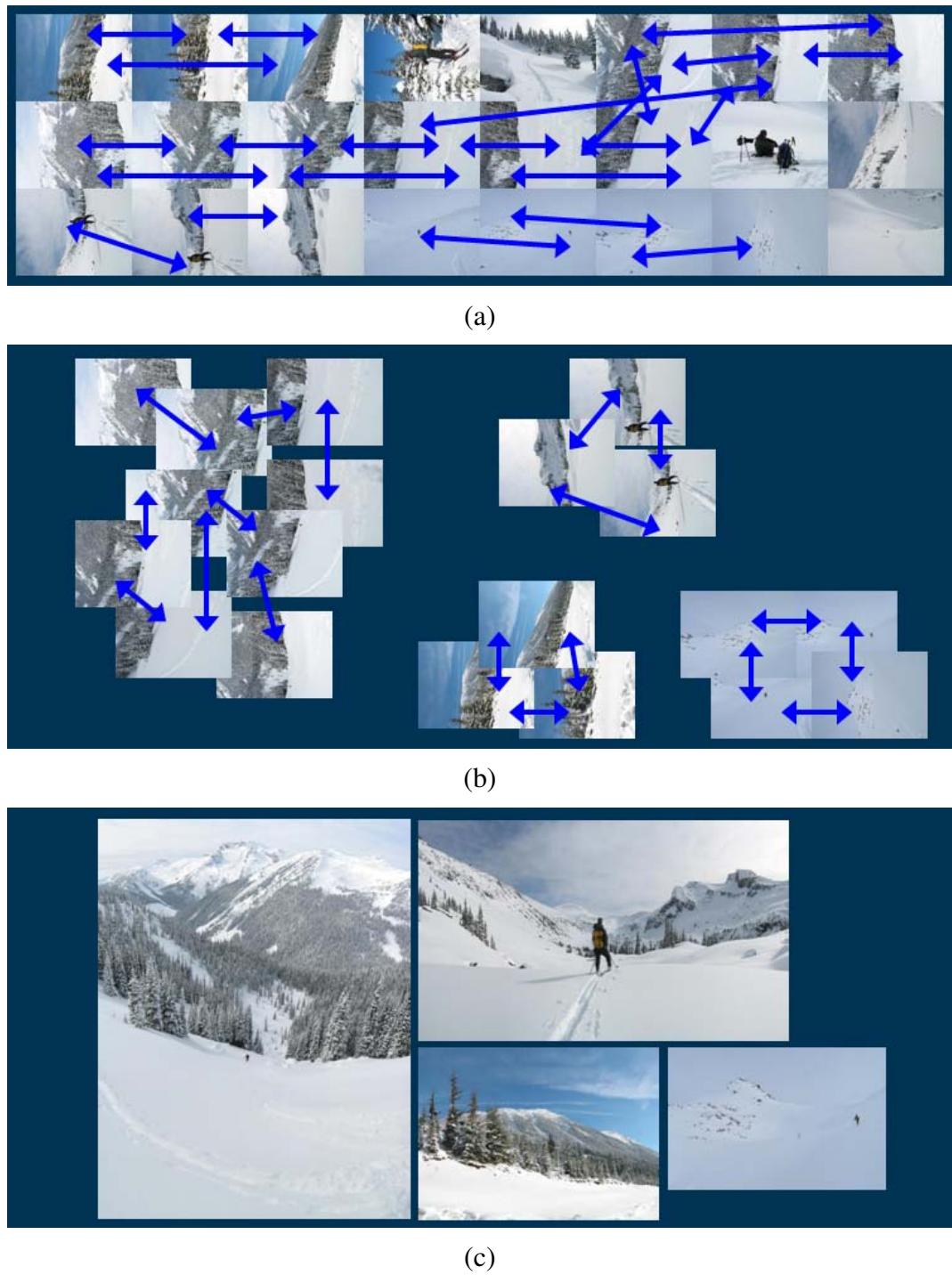


Figure 8.12: *Recognizing panoramas using our new algorithm (Brown et al. 2004)*: (a) input images with pairwise matches; (b) images grouped into connected components (panoramas); (c) individual panoramas registered and blended into stitched composites.

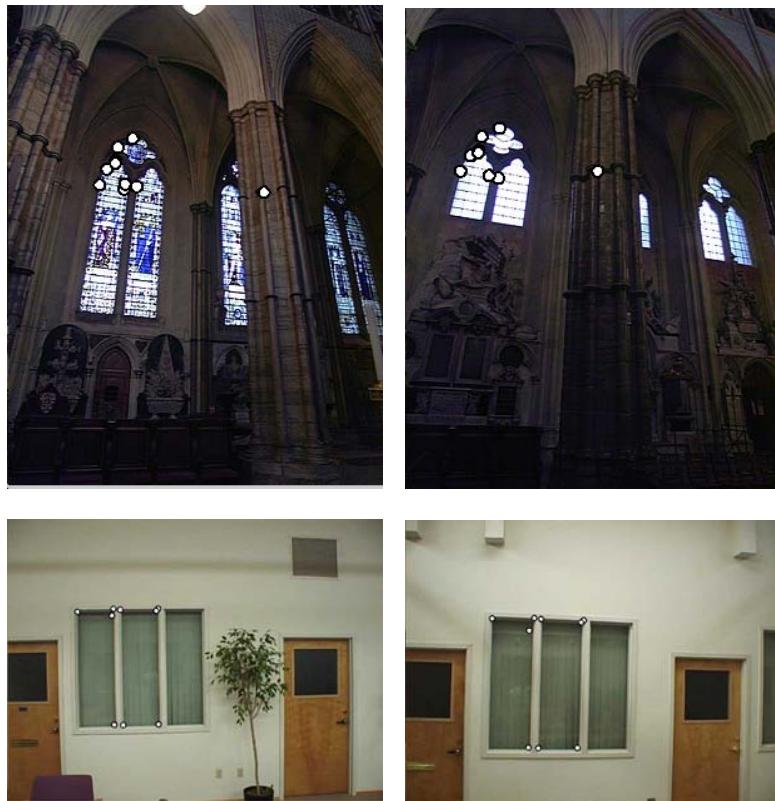


Figure 8.13: *Matching errors (Brown et al. 2004): accidental matching of several features can lead to matches between pairs of images that do not actually overlap.*

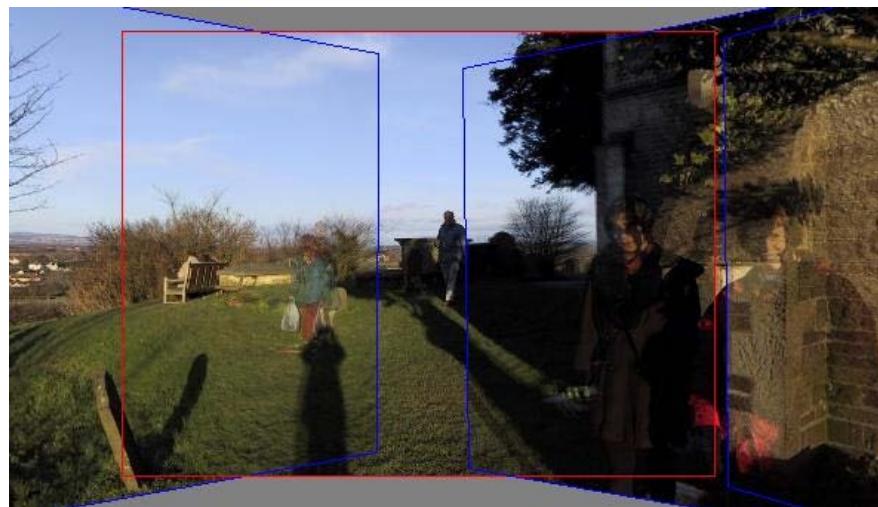


Figure 8.14: *Validation of image matches by direct pixel error comparison can fail when the scene contains moving objects.*

find my proceedings, and look up the panel discussion.] Early feature-based methods seemed to get confused in regions that were either too textured or not textured enough. The features would often be distributed unevenly over the images, thereby failing to match image pairs that should have been aligned. Furthermore, establishing correspondences relied on simple cross-correlation between patches surrounding the feature points, which did not work well when the images were rotated or had foreshortening due to homographies.

Today, feature detection and matching schemes are remarkably robust, and can even be used for known object recognition from widely separated views (Lowe 2004). Features not only respond to regions of high “cornerness” (Förstner 1986, Harris and Stephens 1988), but also to “blob-like” regions (Lowe 2004), as well as uniform areas (Tuytelaars and Van Gool 2004). Furthermore, because they operate in scale-space and use a dominant orientation (or orientation invariant descriptors), they can match images that differ in scale, orientation, and even foreshortening. My own recent experience in working with feature-based approaches is that if the features are well distributed over the image and the descriptors reasonably designed for repeatability, enough correspondences to permit image stitching can usually be found (Brown *et al.* 2005).

The other major reason I used to prefer direct methods was that they make optimal use of the information available in image alignment, since they measure the contribution of *every* pixel in the image. Furthermore, assuming a Gaussian noise model (or a robustified version of it), they properly weight the contribution of different pixels, e.g., by emphasizing the contribution of high-gradient pixels. (See Baker *et al.* (2003a), who suggest that adding even more weight at strong gradients is preferable because of noise in the gradient estimates.) One could argue that for a blurry image with only slowly varying gradients, a direct approach will find an alignment, whereas a feature detector will fail to find anything. However, such images rarely occur in practice in consumer imaging, and the use of scale-space features means that some features can be found at lower resolutions.

The biggest disadvantage of direct techniques is that they have a limited range of convergence. Even though they can be used in a hierarchical (coarse-to-fine) estimation framework, in practice it is hard to use more than two or three levels of a pyramid before important details start to be blurred away. For matching sequential frames in a video, the direct approach can usually be made to work. However, for matching partially overlapping images in photo-based panoramas, they fail too often to be useful. Our older systems for image stitching (Szeliski 1996, Szeliski and Shum 1997) relied on Fourier-based correlation of cylindrical images and motion prediction to automatically align images, but had to be corrected by hand for more complex sequences. Our newer system (Brown *et al.* 2004, Brown *et al.* 2005) uses features and has a good success rate at automatically stitching panoramas without any user intervention.

Is there no rôle then for direct registration? I believe there is. Once a pair of images has been aligned with a feature-based approach, we can warp the two images to a common reference frame and re-compute a more accurate estimate using patch-based alignment. Notice how there is a close

correspondence between the patch-based approximation to direct alignment given in (7.59–7.60) and the inverse covariance weighted feature-based least squares error metric (5.3).

In fact, if we divide the template images up into patches and place an imaginary “feature point” at the center of each patch, the two approaches return exactly the same answer (assuming that the correct correspondences are found in each case). However, for this approach to succeed, we still have to deal with “outliers”, i.e., regions that do not fit the selected motion model due to either parallax (§8.2.2) or moving objects (§8.3.2). While a feature-based approach may make it somewhat easier to reason about outliers (features can be classified as inliers or outliers), the patch-based approach, since it establishes correspondences more densely, is potentially more useful for removing local mis-registration (parallax), as we discuss in §8.2.2.

8.3 Compositing

Once we have registered all of the input images with respect to each other, we need to decide how to produce the final stitched (mosaic) image. This involves selecting a final compositing surface (flat, cylindrical, spherical, etc.) and view (reference image). It also involves selecting which pixels contribute to the final composite and how to optimally blend these pixels to minimize visible seams, blur, and ghosting.

In this section, I review techniques that address these problems, namely compositing surface parameterization, pixel/seam selection, blending, and exposure compensation. My emphasis is on fully *automated* approaches to the problem. Since the creation of high-quality panoramas and composites is as much an *artistic* endeavor as a computational one, various interactive tools have been developed to assist this process, e.g., (Agarwala *et al.* 2004, Li *et al.* 2004b, Rother *et al.* 2004). Some of these are covered in a more detail in the chapter on Computational Photography §9.4.

8.3.1 Choosing a compositing surface

The first choice to be made is how to represent the final image. If only a few images are stitched together, a natural approach is to select one of the images as the *reference* and to then warp all of the other images into the reference coordinate system. The resulting composite is sometimes called a *flat* panorama, since the projection onto the final surface is still a perspective projection, and hence straight lines remain straight (which is often a desirable attribute).

For larger fields of view, however, we cannot maintain a flat representation without excessively stretching pixels near the border of the image. (In practice, flat panoramas start to look severely distorted once the field of view exceeds 90° or so.) The usual choice for compositing larger panoramas is to use a cylindrical (Chen 1995, Szeliski 1996) or spherical (Szeliski and Shum 1997) projection,

as described in §8.1.5. In fact, any surface used for *environment mapping* in computer graphics can be used, including a *cube map* that represents the full viewing sphere with the six square faces of a cube (Greene 1986, Szeliski and Shum 1997). Cartographers have also developed a number of alternative methods for representing the globe (Bugayevskiy and Snyder 1995).

The choice of parameterization is somewhat application dependent, and involves a tradeoff between keeping the local appearance undistorted (e.g., keeping straight lines straight) and providing a reasonably uniform sampling of the environment. Automatically making this selection and smoothly transitioning between representations based on the extent of the panorama is an active area of current research (Kopf *et al.* 2007b).

View selection. Once we have chosen the output parameterization, we still need to determine which part of the scene will be *centered* in the final view. As mentioned above, for a flat composite, we can choose one of the images as a reference. Often, a reasonable choice is the one that is geometrically most central. For example, for rotational panoramas represented as a collection of 3D rotation matrices, we can choose the image whose z -axis is closest to the average z -axis (assuming a reasonable field of view). Alternatively, we can use the average z -axis (or quaternion, but this is trickier) to define the reference rotation matrix.

For larger (e.g., cylindrical or spherical) panoramas, we can still use the same heuristic if a subset of the viewing sphere has been imaged. If the case of full 360° panoramas, a better choice might be to choose the middle image from the sequence of inputs, or sometimes the first image, assuming this contains the object of greatest interest. In all of these cases, having the user control the final view is often highly desirable. If the “up vector” computation described in §8.2.1 is working correctly, this can be as simple as panning over the image or setting a vertical “center line” for the final panorama. [Note: Add a figure here or in §8.2.1 showing the difference before and after up vector estimation.]

Coordinate transformations. Once we have selected the parameterization and reference view, we still need to compute the mappings between the input and output pixels coordinates.

If the final compositing surface is flat (e.g., a single plane or the face of a cube map) and the input images have no radial distortion, the coordinate transformation is the simple homography described by (8.5). This kind of warping can be performed in graphics hardware by appropriately setting texture mapping coordinates and rendering a single quadrilateral.

If the final composite surface has some other analytic form (e.g., cylindrical or spherical), we need to convert every pixel in the final panorama into a viewing ray (3D point) and then map it back into each image according to the projection (and optionally radial distortion) equations. This process can be made more efficient by precomputing some lookup tables, e.g., the partial trigonometric functions needed to map cylindrical or spherical coordinates to 3D coordinates and/or the

radial distortion field at each pixel. It is also possible to accelerate this process by computing exact pixel mappings on a coarser grid and then interpolating these values.

When the final compositing surface is a texture-mapped polyhedron, a slightly more sophisticated algorithm must be used. Not only do the 3D and texture map coordinates have to be properly handled, but a small amount of *overdraw* outside of the triangle footprints in the texture map is necessary, to ensure that the texture pixels being interpolated during 3D rendering have valid values (Szeliski and Shum 1997).

[Note: Steve Seitz's mosaic2.ppt slides discuss some other kinds of geometric surfaces, such as the earth's sphere, as well as other projections, such as slit-scan images (Peleg et al. 2000), concentric mosaics (Shum and He 1999), Steve's unwrapping of a face (Seitz 2001). See the discussion in at the end of this chapter.]

Sampling issues. While the above computations can yield the correct (fractional) pixel addresses in each input image, we still need to pay attention to sampling issues. For example, if the final panorama has a lower resolution than the input images, pre-filtering the input images is necessary to avoid aliasing. These issues have been extensively studied in both the image processing and computer graphics communities. The basic problem is to compute the appropriate pre-filter, which depends on the distance (and arrangement) between neighboring samples in a source image. Various approximate solutions, such as MIP mapping (Williams 1983) or elliptically weighted Gaussian averaging (Greene and Heckbert 1986) have been developed in the graphics community. For highest visual quality, a higher order (e.g., cubic) interpolator combined with a spatially adaptive pre-filter may be necessary (Wang et al. 2001). Under certain conditions, it may also be possible to produce images with a higher resolution than the input images using a process called *super-resolution* (§8.3.4). *[Note: Bill Triggs says that Michael Unser has spline-based resampling that (in some sense) does not need pre-filtering. Add this to the resampling section of the book?]*

8.3.2 Pixel selection and weighting (de-ghosting)

Once the source pixels have been mapped onto the final composite surface, we must still decide how to blend them in order to create an attractive looking panorama. If all of the images are in perfect registration and identically exposed, this is an easy problem (any pixel or combination will do). However, for real images, visible seams (due to exposure differences), blurring (due to mis-registration), or ghosting (due to moving objects) can occur.

Creating clean, pleasing looking panoramas involves both deciding which pixels to use and how to weight or blend them. The distinction between these two stages is a little fluid, since per-pixel weighting can be thought of as a combination of selection and blending. In this section, I

discuss spatially varying weighting, pixel selection (seam placement), and then more sophisticated blending.

Feathering and center-weighting. The simplest way to create a final composite is to simply take an *average* value at each pixel,

$$C(\mathbf{x}) = \sum_k w_k(\mathbf{x}) \tilde{I}_k(\mathbf{x}) / \sum_k w_k(\mathbf{x}), \quad (8.43)$$

where $\tilde{I}_k(\mathbf{x})$ are the *warped* (re-sampled) images and $w_k(\mathbf{x})$ is 1 at valid pixels and 0 elsewhere. On computer graphics hardware, this kind of summation can be performed in an *accumulation buffer* (using the *A* channel as the weight).

Simple averaging usually does not work very well, since exposure differences, mis-registrations, and scene movement are all very visible (Figure 8.15a). If rapidly moving objects are the only problem, taking a *median* filter (which is a kind of pixel selection operator) can often be used to remove them (Irani and Anandan 1998) (Figure 8.15b). Conversely, center-weighting (discussed below) and *minimum likelihood* selection (Agarwala *et al.* 2004) can sometimes be used to retain multiple copies of a moving object (Figure 8.18).

A better approach to averaging is to weight pixels near the center of the image more heavily and to down-weight pixels near the edges. When an image has some cutout regions, down-weighting pixels near the edges of both cutouts and edges is preferable. This can be done by computing a *distance map* or *grassfire transform*,

$$w_k(\mathbf{x}) = \left\| \arg \min_{\mathbf{y}} \{ \|\mathbf{y}\| \mid \tilde{I}_k(\mathbf{x} + \mathbf{y}) \text{ is invalid} \} \right\|, \quad (8.44)$$

where each valid pixel is tagged with its Euclidean distance to the nearest invalid pixel. The Euclidean distance map can be efficiently computed using a two-pass raster algorithm (Danielsson 1980, Borgefors 1986). Weighted averaging with a distance map is often called *feathering* (Szeliski and Shum 1997, Chen and Klette 1999, Uyttendaele *et al.* 2001) and does a reasonable job of blending over exposure differences. However, blurring and ghosting can still be problems (Figure 8.15c). Note that weighted averaging is *not* the same as compositing the individual images with the classic *over* operation (Porter and Duff 1984, Blinn 1994a), even when using the weight values (normalized to sum up to one) as *alpha* (translucency) channels. This is because the over operation attenuates the values from more distant surfaces, and hence is not equivalent to a direct sum.

One way to improve feathering is to raise the distance map values to some large power, i.e., to use $w_k^p(\mathbf{x})$ in (8.43). The weighted averages then become dominated by the larger values, i.e.,

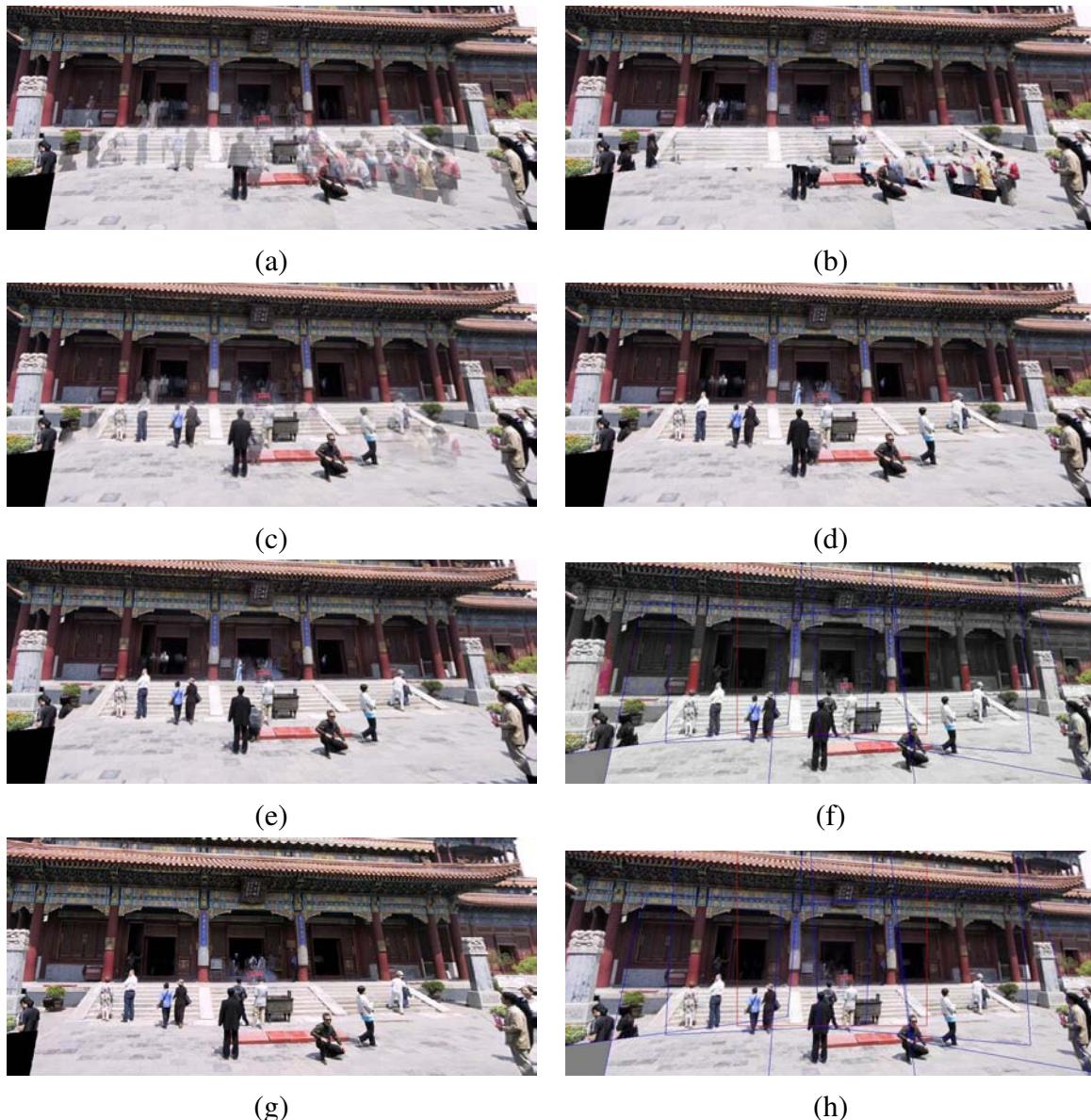


Figure 8.15: *Final composites computed by a variety of algorithms: (a) average, (b) median, (c) feathered average, (d) p-norm $p = 10$, (e) Voronoi, (f) weighted ROD vertex cover with feathering, (g) graph cut seams with Poisson blending, (h) and with pyramid blending.*

they act somewhat like a *p-norm*. The resulting composite can often provide a reasonable tradeoff between visible exposure differences and blur (Figure 8.15d).

In the limit as $p \rightarrow \infty$, only the pixel with the maximum weight gets selected,

$$C(\mathbf{x}) = \tilde{I}_{l(\mathbf{x})}(\mathbf{x}), \quad (8.45)$$

where

$$l = \arg \max_k w_k(\mathbf{x}) \quad (8.46)$$

is the *label assignment* or *pixel selection* function that selects which image to use at each pixel. This hard pixel selection process produces a visibility mask-sensitive variant of the familiar *Voronoi diagram*, which assigns each pixel to the nearest image center in the set (Wood *et al.* 1997, Peleg *et al.* 2000). [Note: Re-read the above two papers to get the facts and terminology right, and check whether to use (Peleg *et al.* 2000) instead of (Peleg and Herman 1997).] The resulting composite, while useful for artistic guidance and in high-overlap panoramas (*manifold mosaics*) tends to have very hard edges with noticeable seams when the exposures vary (Figure 8.15e).

Xiong and Turkowski (1998) use this Voronoi idea (local maximum of the grassfire transform) to select seams for Laplacian pyramid blending (which is discussed below). However, since the seam selection is performed sequentially as new images are added in, some artifacts can occur.

Optimal seam selection. Computing the Voronoi diagram is one way to select the *seams* between regions where different images contribute to the final composite. However, Voronoi images totally ignore the local image structure underlying the seam.

A better approach is to place the seams in regions where the images agree, so that transitions from one source to another are not visible. In this way, the algorithm avoids “cutting through” moving objects where a seam would look unnatural (Davis 1998). For a pair of images, this process can be formulated as a simple dynamic program starting from one (short) edge of the overlap region and ending at the other (Milgram 1975, Milgram 1977, Davis 1998, Efros and Freeman 2001).

When multiple images are being composited, the dynamic program idea does not readily generalize. (For square texture tiles being composited sequentially, Efros and Freeman (2001) run a dynamic program along each of the four tile sides.)

To overcome this problem, Uyttendaele *et al.* (2001) observed that for well-registered images, moving objects produce the most visible artifacts, namely translucent looking *ghosts*. Their system therefore decides which objects to keep and which ones to erase. First, the algorithm compares all overlapping input image pairs to determine *regions of difference* (RODs) where the images disagree. Next, a graph is constructed with the RODs as vertices and edges representing ROD pairs that overlap in the final composite (Figure 8.16). Since the presence of an edge indicates an area of disagreement, vertices (regions) must be removed from the final composite until no edge spans a

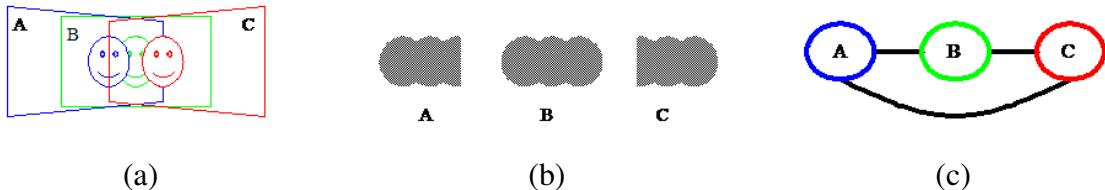


Figure 8.16: *Computation of regions of differences (RODs): (a) three overlapping images with a moving face; (b) corresponding RODs; (c) graph of coincident RODs. (Taken from (Uyttendaele et al. 2001)).*



Figure 8.17: *From a set of five source images (of which four are shown on the left), Photomontage quickly creates a composite family portrait in which everyone is smiling and looking at the camera (right). Users simply flip through the stack and coarsely draw strokes using the designated source image objective over the people they wish to add to the composite. The user-applied strokes and computed regions are color-coded by the borders of the source images on the left (middle). (Copied, with permission, from (Agarwala et al. 2004)).*

pair of remaining vertices. The smallest such set can be computed using a *vertex cover* algorithm. Since several such covers may exist, a *weighted vertex cover* is used instead, where the vertex weights are computed by summing the feather weights in the ROD (Uyttendaele *et al.* 2001). The algorithm therefore prefers removing regions that are near the edge of the image, which reduces the likelihood that partially visible objects will appear in the final composite. (It is also possible to infer which object in a region of difference is the foreground object by the “edginess” (pixel differences) across the ROD boundary, which should be higher when an object is present (Herley 2005).) Once the desired excess regions of difference have been removed, the final composite can be created using a feathered blend (Figure 8.15f).

A different approach to pixel selection and seam placement was recently proposed by Agarwala *et al.* (2004). Their system computes the label assignment that optimizes the sum of two objective functions. The first is a per-pixel *image objective* that determines which pixels are likely to produce



Figure 8.18: *Set of five photos tracking a snowboarder’s jump stitched together into a seamless composite. Because the algorithm prefers pixels near the center of the image, multiple copies of the boarder are retained.*

good composites,

$$\mathcal{C}_D = \sum_{\mathbf{x}} D_l(\mathbf{x})(\mathbf{x}), \quad (8.47)$$

where $D_l(\mathbf{x})(\mathbf{x})$ is the *data penalty* associated with choosing image l at pixel \mathbf{x} . In their system, users can select which pixels to use by “painting” over an image with the desired object or appearance, which sets $D(\mathbf{x}, l)$ to a large value for all labels l other than the one selected by the user (Figure 8.17). Alternatively, automated selection criteria can be used, such as *maximum likelihood* that prefers pixels that occur repeatedly (for object removal), or *minimum likelihood* for objects that occur infrequently (for greatest object retention). Using a more traditional center-weighted data term tends to favor objects that are centered in the input images (Figure 8.18).

The second term is a *seam objective* that penalizes differences in labelings between adjacent images,

$$\mathcal{C}_S = \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{N}} S_{l(\mathbf{x}), l(\mathbf{y})}(\mathbf{x}, \mathbf{y}) \quad (8.48)$$

where $S_{l(\mathbf{x}), l(\mathbf{y})}(\mathbf{x}, \mathbf{y})$ is the image-dependent *interaction penalty* or *seam cost* of placing a seam between pixels \mathbf{x} and \mathbf{y} , and \mathcal{N} is the set of \mathcal{N}_4 neighboring pixels. For example, the simple color-based seam penalty used in (Kwatra *et al.* 2003, Agarwala *et al.* 2004) can be written as

$$S_{l(\mathbf{x}), l(\mathbf{y})}(\mathbf{x}, \mathbf{y}) = \|\tilde{I}_{l(\mathbf{x})}(\mathbf{x}) - \tilde{I}_{l(\mathbf{y})}(\mathbf{x})\| + \|\tilde{I}_{l(\mathbf{x})}(\mathbf{y}) - \tilde{I}_{l(\mathbf{y})}(\mathbf{y})\|. \quad (8.49)$$

More sophisticated seam penalties can also look at image gradients or the presence of image edges (Agarwala *et al.* 2004). Seam penalties are widely used in other computer vision applications such

as stereo matching (Boykov *et al.* 2001) to give the labeling function its *coherence* or *smoothness*. An alternative approach, which places seams along strong consistent edges in overlapping images using a watershed computation has recently been developed by Soille (2006).

The sum of the two objective functions is often called the *Markov Random Field* (MRF) energy, since it arises as the negative log-likelihood of an MRF distribution (Geman and Geman 1984). For general energy functions, finding the minimum can be NP-hard (Boykov *et al.* 2001). However, a variety of approximate optimization techniques have been developed over the years, including *simulated annealing* (Geman and Geman 1984), graph cuts (Boykov *et al.* 2001), and loopy belief propagation (Sun *et al.* 2003, Tappen and Freeman 2003). Both Kwatra *et al.* (2003) and Agarwala *et al.* (2004) use graph cuts, which involves cycling through a set of simpler α -expansion relabelings, each of which can be solved with a graph cut (max-flow) polynomial-time algorithm (Boykov *et al.* 2001).

For the result shown in Figure 8.15g, Agarwala *et al.* (2004) use a large data penalty for invalid pixels and 0 for valid pixels. Notice how the seam placement algorithm avoids regions of differences, including those that border the image and which might result in cut off objects. Graph cuts (Agarwala *et al.* 2004) and vertex cover (Uyttendaele *et al.* 2001) often produce similar looking results, although the former is significantly slower since it optimizes over all pixels, while the latter is more sensitive to the thresholds used to determine regions of difference.

8.3.3 Blending

Once the seams have been placed and unwanted object removed, we still need to blend the images to compensate for exposure differences and other mis-alignments. The spatially-varying weighting (feathering) previously discussed can often be used to accomplish this. However, it is difficult in practice to achieve a pleasing balance between smoothing out low-frequency exposure variations and retaining sharp enough transitions to prevent blurring (although using a high exponent does help).

Laplacian pyramid blending. An attractive solution to this problem was developed by Burt and Adelson (1983b). Instead of using a single transition width, a frequency-adaptive width is used by creating a band-pass (Laplacian) pyramid and making the transition widths a function of the pyramid level. The process is described in more detail in §3.4.4. In practice, a small number of levels (as few as two (Brown and Lowe 2003a)) may be adequate to compensate for differences in exposure. The result of applying this pyramid blending is shown in Figure 8.15i.

Gradient domain blending. An alternative approach to multi-band image blending is to perform the operations in the *gradient domain*. Reconstructing images from their gradient fields has a long



Figure 8.19: *Poisson Image Editing* (Pérez et al. 2003): The dog and the two children on the left are chosen as source images to be pasted into the destination swimming pool. Simple pasting fails to match the colors at the boundaries, whereas Poisson image blending masks these differences.

history in computer vision (Horn 1986), starting originally with work in brightness constancy (Horn 1974), shape from shading (Horn and Brooks 1989), and photometric stereo (Woodham 1981). More recently, related ideas have been used for reconstructing images from their edges (Elder and Goldberg 2001), removing shadows from images (Weiss 2001), separating reflections from a single image (Levin et al. 2004), and *tone mapping* high dynamic range images by reducing the magnitude of image edges (gradients) (Fattal et al. 2002).

Pérez et al. (2003) showed how gradient domain reconstruction can be used to do seamless object insertion in image editing applications (Figure 8.19). Rather than copying pixels, the *gradients* of the new image fragment are copied instead. The actual pixel values for the copied area are then computed by solving a *Poisson equation* that locally matches the gradients while obeying the fixed *Dirichlet* (exact matching) conditions at the seam boundary. Pérez et al. (2003) show that this is equivalent to computing an additive *membrane* interpolant of the mismatch between the source and destination images along the boundary. (The membrane interpolant is known to have nicer interpolation properties for arbitrary-shaped constraints than frequency-domain interpolants (Nielson 1993).) In earlier work, Peleg (1981) also proposed adding a smooth function to force a consistency along the seam curve.

Agarwala et al. (2004) extended this idea to a multi-source formulation, where it no longer makes sense to talk of a destination image whose exact pixel values must be matched at the seam. Instead, each source image contributes its own gradient field, and the Poisson equation is solved using *Neumann* boundary conditions, i.e., dropping any equations that involve pixels outside the

boundary of the image.

Rather than solving the Poisson partial differential equations, Agarwala *et al.* (2004) directly minimize *variational problem*,

$$\min_{C(\mathbf{x})} \|\nabla C(\mathbf{x}) - \nabla \tilde{I}_l(\mathbf{x})(\mathbf{x})\|^2. \quad (8.50)$$

The discretized form of this equation is a set of gradient constraint equations

$$C(\mathbf{x} + \hat{\mathbf{i}}) - C(\mathbf{x}) = \tilde{I}_l(\mathbf{x})(\mathbf{x} + \hat{\mathbf{i}}) - \tilde{I}_l(\mathbf{x})(\mathbf{x}) \text{ and} \quad (8.51)$$

$$C(\mathbf{x} + \hat{\mathbf{j}}) - C(\mathbf{x}) = \tilde{I}_l(\mathbf{x})(\mathbf{x} + \hat{\mathbf{j}}) - \tilde{I}_l(\mathbf{x})(\mathbf{x}), \quad (8.52)$$

where $\hat{\mathbf{i}} = (1, 0)$ and $\hat{\mathbf{j}} = (0, 1)$ are unit vectors in the x and y directions.⁸ They then solve the associated sparse least squares problem. Since this system of equations is only defined up to an additive constraint, Agarwala *et al.* (2004) ask the user to select the value of one pixel. In practice, a better choice might be to weakly bias the solution towards reproducing the original color values.

In order to accelerate the solution of this sparse linear system, (Fattal *et al.* 2002) use multigrid, whereas (Agarwala *et al.* 2004) use hierarchical basis preconditioned conjugate gradient descent (Szeliski 1990b, Szeliski 2006b) §A.5. Even more recently, Agarwala (2007) shows how using a quadtree representation for the solution can further accelerate the computation with minimal loss in accuracy, while Szeliski *et al.* (2008a) show how representing the per-image offset fields using even coarser splines is even faster. This latter work also argues that blending in the log domain, i.e., using multiplicative rather than additive offsets, is preferable, as it more closely matches texture contrasts across seam boundaries. The resulting seam blending work very well in practice (Figure 8.15h), although care must be taken when copying large gradient values near seams so that a “double edge” is not introduced. [*Note: A good test case for properly preconditioned CG is an irregular boundary. Regular multigrid should not work as well.*]

Copying gradients directly from the source images after seam placement is just one approach to gradient domain blending. The paper by Levin *et al.* (2004) examines several different variants on this approach, which they call *Gradient-domain Image STitching* (GIST). The techniques they examine include feathering (blending) the gradients from the source images, as well as using an L1 norm in performing the reconstruction of the image from the gradient field, rather than using an L2 norm as in (8.50). Their preferred technique is the L1 optimization of a feathered (blended) cost function on the original image gradients (which they call GIST1- l_1). Since L1 optimization using linear programming can be slow, they develop a faster iterative median-based algorithm in a multigrid framework. Visual comparisons between their preferred approach and what they call *optimal seam on the gradients* (which is equivalent to Agarwala *et al.* (2004)’s approach) show similar results, while significantly improving on pyramid blending and feathering algorithms.

⁸ At seam locations, the right hand side is replaced by the average of the gradients in the two source images.

Exposure compensation. Pyramid and gradient domain blending can do a good job of compensating for moderate amounts of exposure differences between images. However, when the exposure differences become large, alternative approaches may be necessary.

Uyttendaele *et al.* (2001) iteratively estimate a local correction between each source image and a blended composite. First, a block-based quadratic transfer function is fit between each source image and an initial feathered composite. Next, transfer functions are averaged with their neighbors to get a smoother mapping, and per-pixel transfer functions are computed by *splining* (interpolating) between neighboring block values. Once each source image has been smoothly adjusted, a new feathered composite is computed, and the process is repeated (typically 3 times). The results in (Uyttendaele *et al.* 2001) demonstrate that this does a better job of exposure compensation than simple feathering, and can handle local variations in exposure due to effects like lens vignetting.

[Note: Mention (Eden *et al.* 2006) and give a forward pointer to §9.2 HDR, where instead of compensating for exposures, true radiance is first recovered and then tone mapped (the right thing to do).]

8.3.4 Application: Photomontage

[Note: Describe how PhotoMontage puts the user in the loop to select the best expression or content. Make this an exercise.]

Extensions and open issues (incorporate and remove section)

In this paper, I have surveyed the basics of image alignment and stitching, concentrating on techniques for registering partially overlapping images and blending them to create seamless panoramas. A large number of additional techniques have been developed for solving related problems such as increasing the resolution of images by taking multiple displaced pictures (*super-resolution*), stitching videos together to create dynamic panoramas, and stitching videos and images in the presence of large amounts of parallax.

Perhaps the most common question that comes up in relation to image stitching is the following. “Why not just take multiple images of the same scene with sub-pixel displacements and produce an image with a higher effective resolution?” Indeed, this problem has been studied for a long time and is generally known as *multiple image super-resolution*.⁹ Examples of papers that have addressed this issue include (Keren *et al.* 1988, Irani and Peleg 1991, Cheeseman *et al.* 1993, Capel and Zisserman 1998, Capel and Zisserman 2000, Chaudhuri 2001). [Note: Here are more

⁹ One can also increase the resolution of a single image using various kinds of non-linear or example-based interpolation techniques (Freeman *et al.* 2002, Baker and Kanade 2002).

reference, (Keren et al. 1988, Irani and Peleg 1991, Cheeseman et al. 1993, Mann and Picard 1994, Chiang and Boult 1996, Basile et al. 1996, Capel and Zisserman 1998, Smelyanskiy et al. 2000, Capel and Zisserman 2000, Capel and Zisserman 2001, Chaudhuri 2001). and Simon Baker's PAMI paper (Baker and Kanade 2002) has even more.] (See (Baker and Kanade 2002) for a recent paper with lots of additional references and experimental comparisons.) The general idea is that different images of the same scene taken from slightly different positions (i.e., where the pixels do not sample exactly the same rays in space) contain more information than a single image. However, this is only true if the imager actually *aliases* the original signal, e.g., if the silicon sensor integrates over a finite area and the optics do not cut off all the frequencies above the Nyquist frequency. Motion estimation also needs to be very accurate for this to work, so that in practice, an increase in resolution greater than $2\times$ is difficult to achieve (Baker and Kanade 2002).

Another popular topic is video stitching (Teodosio and Bender 1993, Massey and Bender 1996, Sawhney and Ayer 1996, Irani and Anandan 1998, Baudisch et al. 2005, Steedly et al. 2005). While this problem is in many ways a straightforward generalization of multiple-image stitching, the potential presence of large amounts of independent motion, camera zoom, and the desire to visualize dynamic events impose additional challenges. For example, moving foreground objects can often be removed using *median filtering*. Alternatively, foreground objects can be extracted into a separate layer (Sawhney and Ayer 1996) and later composited back into the stitched panoramas, sometimes as multiple instances to give the impressions of a “Chronophotograph” (Massey and Bender 1996) and sometimes as video overlays (Irani and Anandan 1998). [Note: adding temporal elements (Sarnoff’s mosaics with video (*Lamplight?*)] Videos can also be used to create animated *panoramic video textures* in which different portions of a panoramic scene are animated with independently moving video loops (Agarwala et al. 2005, Rav-Acha et al. 2005).

Video can also provide an interesting source of content for creating panoramas taken from moving cameras. While this invalidates the usual assumption of a single point of view (optical center), interesting results can still be obtained. For example the VideoBrush system (Sawhney et al. 1998) uses thin strips taken from the center of the image to create a panorama taken from a horizontally moving camera. This idea can be generalized to other camera motions and compositing surfaces using the concept of mosaics on adaptive manifold (Peleg et al. 2000), and also used to generate panoramic stereograms (Peleg and Ben-Ezra 1999). Related ideas have been used to create panoramic matte paintings for multi-plane cell animation (Wood et al. 1997), for creating stitched images of scenes with parallax (Kumar et al. 1995), and as 3D representations of more complex scenes using *multiple-center-of-projection images* (Rademacher and Bishop 1998).

Another interesting variant on video-based panoramas are *concentric mosaics* (Shum and He 1999). Here, rather than trying to produce a single panoramic image, the complete original video is kept and used to re-synthesize novel views (from different camera origins) using ray remapping (light field rendering), thus endowing the panorama with a sense of 3D depth. The same data set

can also be used to explicitly reconstruct the depth using multi-baseline stereo (Shum and Szeliski 1999, Shum *et al.* 1999, Peleg *et al.* 2001, Li *et al.* 2004a).

[Note: Other applications: document scanning with a mouse (Nakao *et al.* 1998); retinal image mosaics (Can *et al.* 2002).]

Open issues. While image stitching is by now a fairly mature field with a variety of commercial products, there remain a large number of challenges and open extensions. One of these is to increase the reliability of fully automated stitching algorithms. As discussed in §8.2.3 and illustrated in Figures 8.13 and 8.14, it is difficult to simultaneously avoid matching spurious features or repeated patterns while also being tolerant to large outliers such as moving people. Advances in semantic scene understanding could help resolve some of these problems, as well as better machine learning techniques for feature matching and validation.

The problem of parallax has also not been adequately solved. For small amounts of parallax, the deghosting techniques described in §8.2.2 and §8.3.2 can often adequately disguise these effects through local warping and careful seam selection. For high-overlap panoramas, concentric mosaics *concentric mosaics* (Shum and He 1999), panoramas with parallax (Li *et al.* 2004a) and careful seam selection (with potential user guidance) (Agarwala *et al.* 2004) can be used. The most challenging case is limited overlap panoramas with large parallax, since the depth estimates needed to compensate for the parallax are only available in the overlap regions (Kang *et al.* 2004, Uyttendaele *et al.* 2004).

[Note: How to really make these things work automatically: repeated pattern, matching subsets, moving objects, parallax. Hard to get the last 3%. (Mention internal test data suite, shipping in product.)

Automated object removal: like intelligent PhotoMontage (semantic stitching, photographer's assistant)

Large parallax: need to do 3D reconstruction. But, not possible if no overlap in some regions (MPPS gets around this with a hack). Ideally, want 70% overlap to tie inter-frame motions strongly together (also for better blending). Video-rate cameras with on-board stitching may some day solve this...]

8.4 Exercises

Ex 8.1 (Alignment-based mosaics) Take a pair of images, compute a coarse-to-fine alignment (after initial manual or programmatic positioning), blend the result. Align additional images to the mosaic.

Ex 8.2 (Featured-based mosaics) Extract features, do long-range matching, compute an alignment. Optional: make your technique more robust. Blend as above, add more matches (either tracking through time, or doing pairwise with the previous image).

Ex 8.3 (Blending and feathering) Compute a feather (distance) map and use it to blend. Try different variations on the blend function. Optional: use Laplacian pyramids (Exercise 3.20).

Ex 8.4 (Coarse alignment) Use FFT or phase correlation to estimate the initial alignment between successive images. How well does this work? Over what range of overlaps? If it does not work, does aligning sub-sections (e.g., quarters) do better?

Ex 8.5 (Automated mosaicing) Determine the alignment between all pairs of images. Apply the coarse alignment to any pair of images, and return a measure of likelihood of overlap (see (Schafalitzky and Zisserman 2002, Brown and Lowe 2003a, Sivic and Zisserman 2003)).

Completely automated mosaic: unknown order, arrangement, focal lengths, orientation; use an orientation invariant feature matcher, figure out orientation, then bundle adjust away... (may need RANSAC)

Ex 8.6 (Global alignment) Use a feature-based technique (or patch-based alignment) to seed a bundle adjuster. Use a pure rotational model, or add a guessed translation to each camera position.

After bundle adjustment, use the camera positions to generate an arbitrary new view.

Ex 8.7 (Environment mapping) Take a geometric object (e.g. cube) and construct its environment map.

Ex 8.8 (De-ghosting) Use the results of the previous bundle adjustment to predict the location of each feature (or patch) in a consensus geometry. Take the predicted feature location, using either a central camera, or ...?

Use multi-way morph to do the rendering.

Ex 8.9 (Pyramid vs. gradient domain blending) Is there a relationship between Laplacian pyramid and gradient domain blending?

Try reasoning about how different frequencies are treated by the two techniques.

Is there a way to design a hybrid algorithm that has the best feature (what would they be?) from both?

Ex 8.10 (Photomontage and object removal) Semantically meaningful object removal: deal with ghosts the “right” way (may want user preferences, or UI).

Implement the PhotoMontage system with user strokes or regions (like in GroupShot).

[Note: Try to identify some open research problems for graduate students to work on...]

Chapter 9

Computational photography

9.1	Photometric calibration	410
9.1.1	Radiometric response function.	410
9.1.2	Vignetting	414
9.1.3	Optical blur (spatial response estimation)	416
9.2	High dynamic range imaging	419
9.2.1	Tone mapping	427
9.2.2	<i>Application:</i> Flash photography	435
9.3	Super-resolution and blur removal	437
9.3.1	Color image de-mosaicing	438
9.4	Image matting and compositing	439
9.4.1	Blue screen matting	440
9.4.2	Natural image matting	442
9.4.3	Optimization-based matting	446
9.5	Texture analysis and synthesis	451
9.5.1	<i>Application:</i> Hole filling and inpainting	451
9.6	Non-photorealistic rendering	451
9.7	Exercises	452

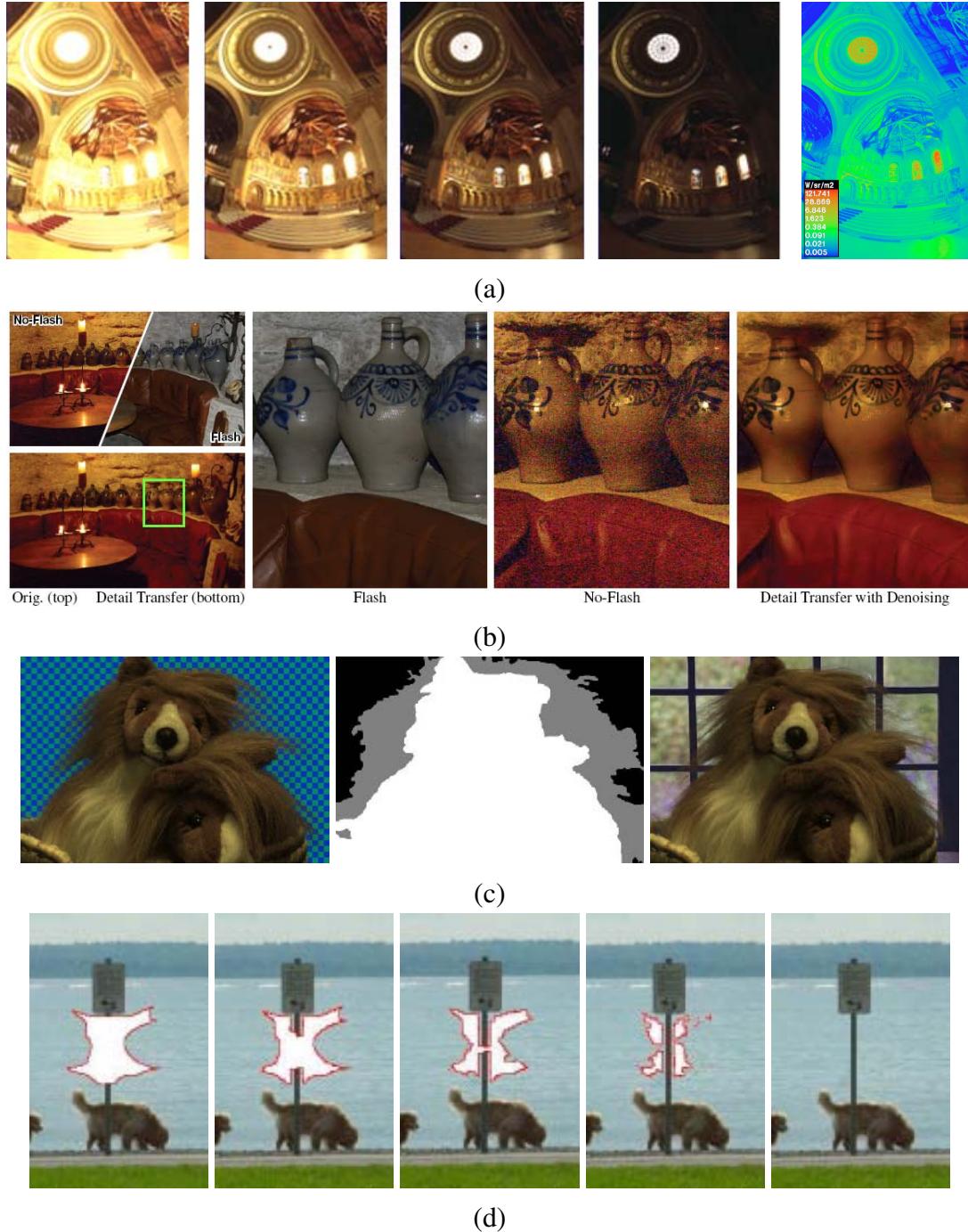


Figure 9.1: Some examples of computational photography: (a) merging multiple exposures to create high dynamic range images ([Debevec and Malik 1997](#)); (b) merging flash and non-flash photographs; ([Petschnigg et al. 2004](#)); (c) image matting and compositing; ([Chuang et al. 2001](#)); (d) hole filling with inpainting ([Criminisi et al. 2004](#)).

Stitching multiple images into wide field of view panoramas, which we covered in the previous chapter, allows us create photographs that could not be captured with a regular camera. This is just one instance of *computational photography*, where image analysis and processing algorithms are applied to one or more photographs to create novel images that go beyond the capabilities of traditional imaging systems.

In this chapter, I cover a number of additional computational photography algorithms. I begin with a review of photometric image calibration §9.1, i.e., the measurement of camera and lens responses, which is a prerequisite for many of the algorithms I describe later. I then discuss *high dynamic range imaging* §9.2, which captures the full range of brightnesses in a scene through the use of multiple exposures (Figure 9.1a). I also discuss *tone mapping operators* that map such rich images back into regular display devices such as screens and printers, as well as algorithms that merge flash and regular images to obtain better exposures (Figure 9.1b).

Next, I discuss how the resolution of images can be improved either by merging multiple photographs together or using sophisticated image priors §9.3. This includes algorithms for extracting full color images from the patterned Bayer mosaics present in most cameras.

I then discuss algorithms for cutting pieces of images from one photograph and pasting them into other ones §9.4 (Figure 9.1c), and how to generate novel textures from real-world samples for applications such as filling holes in images §9.5 (Figure 9.1d). I close with a brief overview of *non-photorealistic rendering* §9.6, which can turn regular photographs into artistic rendering that resemble traditional drawings and paintings.

One topic that I do not cover extensively in this book are novel computational sensors, optics, and cameras. A nice survey can be found in (Nayar 2006), an upcoming book by Raskar and Tumblin (2009), and some more recent research papers, e.g., (Levin *et al.* 2007). Some related discussion can also be found in the sections on high dynamic range imaging §9.2 and lightfields §12.3.4.

A good general-audience introduction to computational photography can be found in the article by Hayes (2008) as well as survey papers by Nayar (2006), Cohen and Szeliski (2006), Levoy (2006), andDebevec (2006). (See the two special issues edited by Bimber (2006) and Durand and Szeliski (2007).) A forthcoming book on computational photography by Raskar and Tumblin (2009) has extensive coverage of topics in this area, with particular emphasis on computational cameras and sensors. The sub-field of high dynamic range imaging has its own book summarizing research in this area (Reinhard *et al.* 2005), as well as a wonderful book aimed more at professional photographers (Freeman 2008). A good survey on image matting can be found in (Wang and Cohen 2007a).

There are also several courses on Computation Photography where the authors have pro-

vided extensive on-line materials, e.g., Frédo Durand's Computation Photography course at MIT,¹ Alyosha Efros' class at Carnegie Mellon,² Marc Levoy's class at Stanford,³ and a series of SIGGRAPH courses on Computational Photography.⁴

[Note: Have another look at the URLs in Fredo's course notes.]

9.1 Photometric calibration

Before we can successfully merge multiple photographs together, we need to characterize the functions that map incoming irradiance into pixel values and also the amounts of noise present in each image. In this section, I examine three components of the imaging pipeline (Figure 9.2) that affect this mapping.

The first is the *radiometric response function* (Mitsunaga and Nayar 1999), which maps photons arriving at the lens into digital values stored in the image file §9.1.1. The second is *vignetting*, which darkens pixel values near the periphery of images, especially at large apertures §9.1.2. The third is the *point spread function*, which characterizes the blur induced by the lens, anti-aliasing filters, and finite sensor areas §9.1.3. The material in this section builds on the image formation processes described in §2.2.3–2.3.3, so if it's been a while since you looked at those sections, please go back and review them.

9.1.1 Radiometric response function.

As we can see in Figure 9.2, a number of factors affect how the intensity of light arriving at the lens ends up being mapped into stored digital values. Let us ignore for the moment any non-uniform attenuation that may occur inside the lens, which I will cover in §9.1.2.

The first factors to affect this mapping are the aperture and shutter speed §2.3, which can be modeled as global multipliers on the incoming light, most conveniently measured in *exposure values* (\log_2 brightness ratios). Next, the analog to digital (A/D) converter on the sensing chip applies an electronic gain, usually controlled by the ISO setting on your camera. While in theory this gain is linear, as with any electronics, non-linearities may be present (either unintentionally or by design). Ignoring for now the noise introduced by photon noise, on-chip noise, amplifier noise and quantization noise, which I will discuss shortly, you can often assume that the mapping between incoming light and the values stored in a RAW camera file (if your camera supports this) is roughly linear.

¹ MIT 6.815/6.865, <http://stellar.mit.edu/S/course/6/sp08/6.815/materials.html>.

² CMU 15-463, http://graphics.cs.cmu.edu/courses/15-463/2008_fall/.

³ Stanford CS 448A, <http://graphics.stanford.edu/courses/cs448a-08-spring/>.

⁴ <http://web.media.mit.edu/~raskar/photo/>.

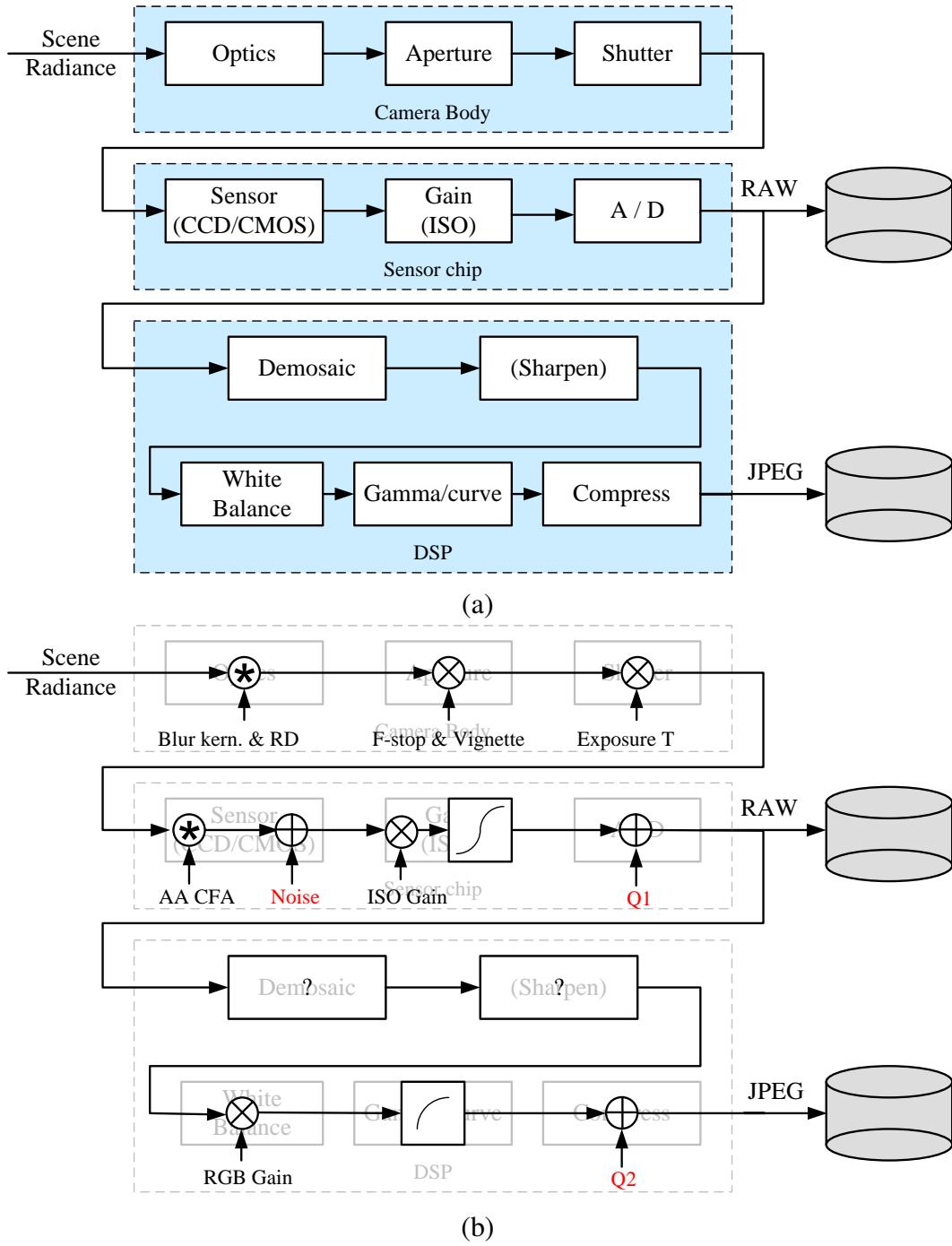


Figure 9.2: *Image sensing pipeline: (a) block diagram showing the various sources of noise as well as the typical digital post-processing steps; (b) equivalent signal transforms, including convolution, gain, and noise injection.*

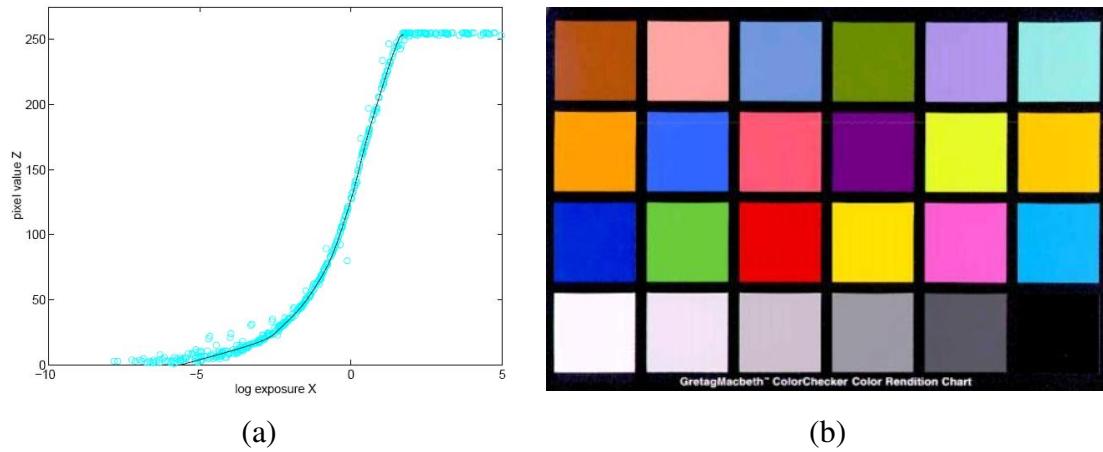


Figure 9.3: *Radiometric response calibration*. (a) A typical camera response function, showing the mapping, for one color channel, between incoming log irradiance (exposure) and output 8-bit pixel values (Debevec and Malik 1997). (b) A color checker chart.

If images are being stored in the more common JPEG format, the camera's digital signal processor (DSP) next performs Bayer pattern de-mosaicing (§2.3.2 and §9.3.1), which is a mostly linear (but often non-stationary) process. Some sharpening is also often applied at this stage. Next, the color values are multiplied by different constants (or sometimes a 3×3 color twist matrix) to perform color balancing, i.e., to move the whitepoint closer to pure white. Finally, a standard gamma is applied to the intensities in each color channel, and the colors are converted into YCbCr format before being transformed by a DCT, quantized, and then compressed into the JPEG format §2.3.3. Figure 9.2 shows all of these steps in pictorial form.

Given the complexity of all of this processing, it is difficult to model the camera response function (Figure 9.3a), i.e., the mapping between incoming irradiance and digital RGB values, from first principles. A more practical approach is to calibrate the camera by measuring correspondences between incoming light and final values.

The most accurate, but most expensive, approach is to use an *integrating sphere*, which is a large (typically 1m) sphere carefully painted on the inside with white matte paint. An accurately calibrated light at the top controls the amount of radiance inside the sphere (which is constant everywhere because of the sphere's radiometry), and a small opening at the side allows for a camera/lens combination to be mounted. By slowly varying the current going into the light, an accurate correspondence can be established between incoming radiance and measured pixel values. The vignetting and noise characteristics of the camera can also be simultaneously determined.

A more practical alternative is to use a calibration chart (Figure 9.3b) such as the Macbeth

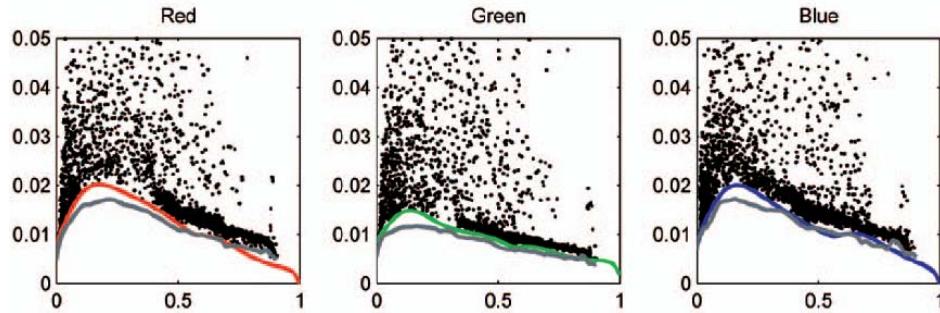


Figure 9.4: *Noise level function estimates obtained from a single color photograph (Liu et al. 2008).* The colored curves are the estimated NLF fit as the probabilistic lower envelope of the measured deviations between the noisy piecewise-smooth images. The ground truth NLFs obtained by averaging 29 images are shown in gray.

or Munsell ColorChecker Chart.⁵ The biggest problem with this approach is to ensure uniform lighting. One approach is to use a large dark room with a high quality light source far away from (and perpendicular to) the chart. Another is to place the chart outdoors away from any shadows. (The results will differ under these two conditions, because the color of the illuminant will be different).

The easiest approach is probably to take multiple exposures of the same scene while the camera is on a tripod, and to recover the response function by simultaneously estimating the incoming irradiance at each pixel and the response curve (Mann and Picard 1995,Debevec and Malik 1997, Mitsunaga and Nayar 1999). This approach is discussed in more detail in section §9.2 on high dynamic range imaging.

If all else fails, i.e., you just have one or more unrelated photos, you can use an International Color Consortium (ICC) profile for the camera. [Note: find a reference: (Fairchild 2005) or http://www.color.org/info_profiles2.xalter] Even more simply, you can just assume that the response is linear if they are RAW files, and that the images have a $\gamma = 2.2$ non-linearity (plus clipping) applied to each RGB channel if they are JPEG images.

Noise level estimation

In addition to knowing the camera response function, it is also often important to know the amount of noise being injected under a particular camera setting (e.g., ISO/gain level). The simplest characterization of noise is a single standard deviation, usually measured in gray levels, independent of pixel value. A more accurate characterization is to estimate the noise level as a function of pixel value (Figure 9.4), which is known as the *noise level function* (Liu et al. 2008).

⁵ <http://www.xrite.com>

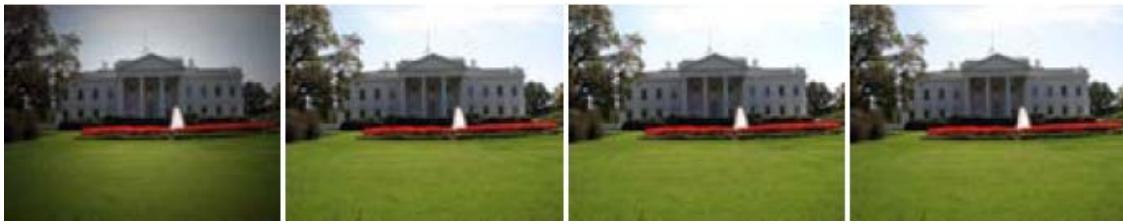


Figure 9.5: *Single image vignetting correction* (Zheng et al. 2008): (a) original image with strong visible vignetting; (b) vignetting compensation from (Zheng et al. 2006); (c–d) vignetting compensation from (Zheng et al. 2008).

As with the camera response function, the simplest way to estimate these quantities is in the lab, using either an integrating sphere or a calibration chart. The noise can either be estimated at each pixel independently by taking repeated exposures and computing the temporal variance in the measurements (Healey and Kondepudy 1994), or over regions by assuming that pixel values should all be the same within some region, e.g., the inside of a color checker square, and computing a spatial variance.

This approach can be generalized to photos where there are regions of constant or slowly varying intensity (Liu et al. 2008). First, segment the image into such regions and fit a constant or linear function inside each region. Next, measure the (spatial) standard deviation of the differences between the noisy input pixels and the smooth fitted function away from large gradients and region boundaries. Plot these as a function of output level for each color channel, as shown in Figure 9.4. Finally, fit a lower envelope to this distribution in order to ignore pixels or deviations that are outliers. A fully Bayesian approach to this problem that models the statistical distributions of each quantity is presented in (Liu et al. 2008). A simpler approach, which should produce useful results in most cases, is to simply fit a low-dimensional function (e.g., positive valued B-spline) to the lower envelope (see Exercise 9.2).

In more recent work, Matsushita and Lin (2007) present a technique for simultaneously estimating a camera’s response and noise level functions. Their paper also contains extensive references to previous work in these areas.

9.1.2 Vignetting

A common problem with using wide angle and/or wide aperture lenses is that the image tends to darken in the corners (Figure 9.5a). This problem is generally known as *vignetting* and comes in several different forms, including natural, optical, and mechanical vignetting §2.2.3 (Ray 2002). As with radiometric response function calibration, the most accurate way to calibrate vignetting is to use an integrating sphere or a picture of a uniformly colored and illuminated blank wall.



Figure 9.6: *Simultaneous estimation of vignetting, exposure, and radiometric response* (Goldman and Chen 2005): (a) original average of the input images; (b) after compensating for vignetting; (c) using gradient domain blending only (note the remaining mottled look); (d) after both vignetting compensation and blending.

An alternative approach is to stitch a panoramic scene and to assume that the true radiance at each pixel comes from the central portion of each input image. This is easier to do if the radiometric response function is already known (e.g., by shooting in RAW mode) and if the exposure is kept constant. If the response function, image exposures, and vignetting function are unknown, they can still be recovered by optimizing a large least squares fitting problem (Litvinov and Schechner 2005, Goldman and Chen 2005). Figure 9.6 shows an example of simultaneously estimating the vignetting, exposure, and radiometric response function from a set of overlapping photographs (Goldman and Chen 2005). Note that unless vignetting is modeled and compensated, regular gradient-domain image blending §8.3.3 (8.50) cannot create an attractive image.

If only a single image is available, vignetting can be estimated by looking for slow consistent intensity variations in the radial direction. The original algorithm proposed by Zheng *et al.* (2006) first pre-segmented the image into smoothly varying regions and then performed an analysis inside each region. Instead of pre-segmenting the image, Zheng *et al.* (2008) compute the radial gradients at all the pixels, and use the asymmetry in this distribution (since gradients away from the center will on average be slightly negative) to estimate the vignetting. Figure 9.5 shows the results of applying both of these algorithms to an image with a large amount of vignetting. Exercise 9.3 has you implement some of the above techniques.

9.1.3 Optical blur (spatial response estimation)

One final characteristic of imaging systems that you should calibrate is the spatial response function, which encodes the optical blur that gets convolved with the incoming image to produce the point-sampled image. The shape of the convolution kernel, which is also known as *point spread function* or *optical transfer function*, depends on several different factors, including lens blur and radial distortion §2.2.3, anti-aliasing filters in front of the sensor, and the shape and extent of each active pixel area §2.3 (Figure 9.2). A good estimate of this function is required for applications such as multi-image super-resolution and de-blurring §9.3.

In theory, one could estimate the PSF by simply observing an infinitely small point light source everywhere in the image. Creating such an array of samples by drilling through a dark plate and backlighting with a very bright light source is difficult in practice.

A more practical approach is to observe an image composed of long straight lines or bars, since these can be fitted to arbitrary precision. Because the location of a horizontal or vertical edge can be *aliased* during acquisition, slightly slanted edges are preferred. The profile and locations of such edges can be estimated to sub-pixel precision, which makes it possible to estimate the PSF at sub-pixel resolutions (Reichenbach *et al.* 1991, Burns and Williams 1999, Williams and Burns 2001, Goesele *et al.* 2003). The thesis by Murphy (2005) contains a nice survey of all aspects of camera calibration, including the spatial frequency response (SFR), spatial uniformity, tone reproduction, color reproduction, noise, dynamic range, color channel registration, and depth of field. It also includes a description of a slant-edge calibration algorithm called `sfrm2`.

The slant-edge technique can be used to recover a 1-D projection of the 2-D PSF, e.g., slightly vertical edges are used to recover the horizontal *line spread function* (LSF) (Williams 1999). The LSF can then be converted into the Fourier domain and its magnitude plotted as a one-dimensional *modulation transfer function* (MTF), which can indicate which image frequencies are lost (blurred) and aliased during the acquisition process §2.3.1. For most computational photography applications, it is preferable to directly estimate the full 2-D PSF, since it can be hard to recover from its projections (Williams 1999).

Figure 9.7 shows a pattern containing edges at all orientations, which can be used to directly recover a two-dimensional PSF. First, corners in the pattern are located by extracting edges in the sensed image, linking them, and finding the intersections of the circular arcs. Next, the ideal pattern, whose analytic form is known, is warped (using a homography) to fit the central portion of the input image, and its intensities are adjusted to fit the ones in the sensed image. If desired, the pattern can be rendered at a higher resolution than the input image, which enables estimating the PSF to sub-pixel resolution (Figure 9.8a). Finally a large linear least-squares system is solved to recover the unknown PSF kernel K ,

$$K = \arg \min_K \|B - D(I * K)\|^2, \quad (9.1)$$

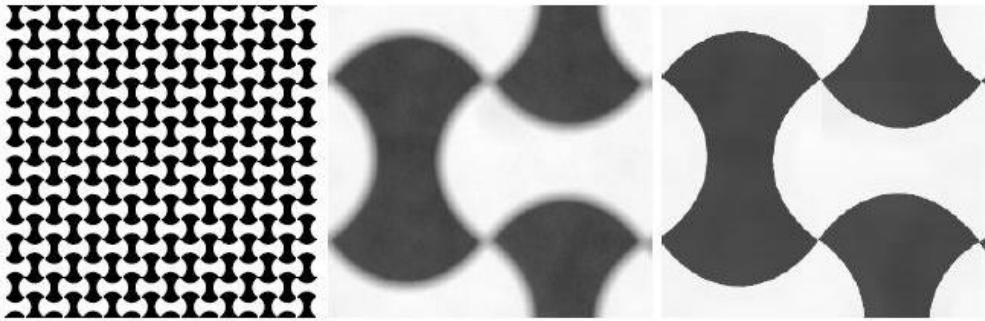


Figure 9.7: *Sample calibration pattern with edge equally distributed at all orientations, which can be used for PSF and radial distortion estimation (Joshi et al. 2008). A closeup of the ideal pattern is shown on the right, and a portion of an actual sensed image in the middle.*

[Note: Replace this with originals from LaTeX source directory.]

where B is the sensed (blurred) image, I is the predicted (sharp) image, and D is an optional downsampling operator that matches the resolution of the ideal and sensed images (Joshi *et al.* 2008).

If the process of estimating the PSF is done locally in overlapping patches of the image, it can also be used to estimate the radial distortion and chromatic aberration induced by the lens (Figure 9.8b). Because the homography mapping the ideal target to the sensed image is estimated in the central (undistorted) part of the image, any (per-channel) shifts induced by the optics manifest themselves as a displacement in the PSF centers.⁶ Compensating for these shifts eliminates both the achromatic radial distortion and the inter-channel shifts that result in visible chromatic aberration. The color-dependent blurring caused by chromatic aberration (Figure 2.22) can also be removed using the de-blurring techniques discussed in §9.3. Figure 9.8b shows how the radial distortion and chromatic aberration manifest themselves as elongated and displaced PSFs, along with the result of removing these effects in a region of the calibration target.

The local 2-D PSF estimation technique can also be used to estimate vignetting. Figure 9.8c shows how the mechanical vignetting manifests itself as clipping of the PSF in the corners of the image. In order for the overall dimming associated with vignetting to be properly captured, the modified intensities of the ideal pattern need to be extrapolated from the center, which is best done with a uniformly illuminated target.

When working with RAW Bayer-pattern images, the correct way to estimate the PSF is to only evaluate the least squares terms in (9.1) at sensed pixel values, while interpolating the ideal image

⁶ This process confounds the distinction between geometric and photometric calibration. In principle, any geometric distortion could be modeled by spatially varying displaced PSFs. In practice, it's easier to fold any large shifts into the geometric correction component.

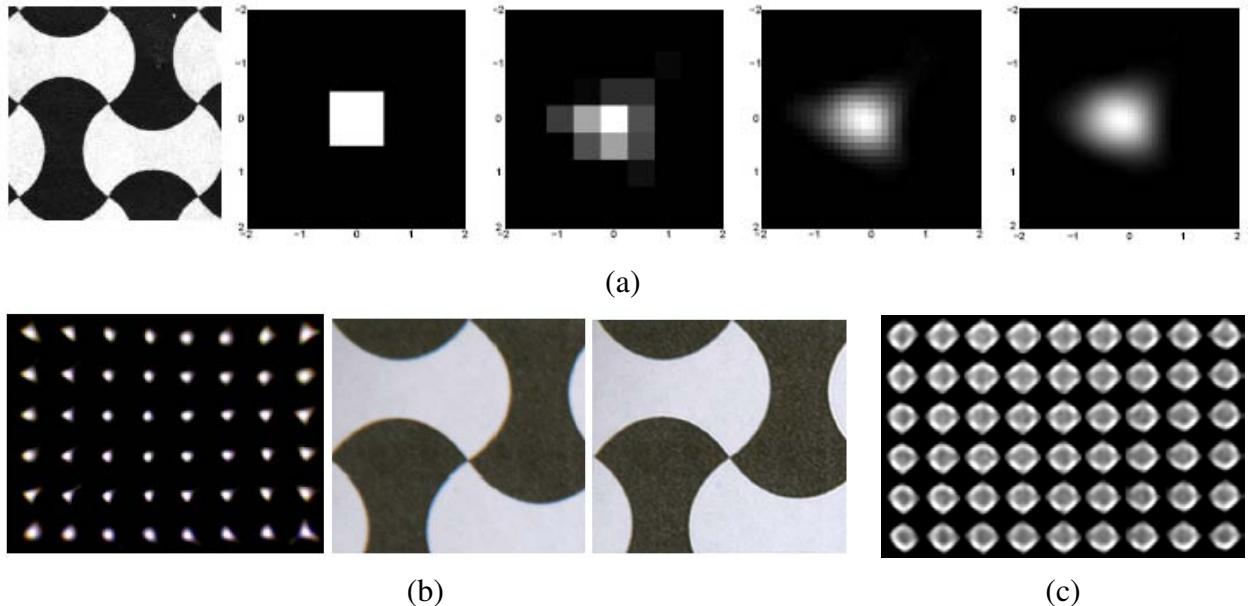


Figure 9.8: *Point spread function estimation using a calibration target (Joshi et al. 2008).* (a) PSFs at successively higher resolutions (note the interaction between the square sensing area and the circular lens blur). (b) The radial distortion and chromatic aberration can also be estimated and removed. (c) PSF for a mis-focused (blurred) lens showing some diffraction and vignetting effects in the corners.

[Note: Replace this with originals from LaTeX source directory and optionally re-do the labels.]

to all values. For JPEG images, you should linearize your intensities first, e.g., remove the gamma and any other non-linearities in your estimated radiometric response function.

What if you have an image that was taken with an uncalibrated camera? Can you still recover the PSF and use it to correct the image? In fact, with a slight modification, the previous algorithms still work.

Instead of assuming a known calibration image, you can detect strong elongated edges and fit ideal step edges in such regions (Figure 9.9b), resulting in the sharp image shown in Figure 9.9d. For every pixel that is surrounded by a complete set of valid estimated neighbors (green pixels in Figure 9.9c), apply the least squares formula (9.1) to estimate the kernel K . The resulting locally estimated PSFs can be used to correct for chromatic aberration (since the relative displacements between per-channel PSFs can be computed), as shown in (Joshi et al. 2008).

Exercise 9.4 provides some more detailed instructions for implementing and testing edge-based PSF estimation algorithms. An alternative approach, which does not require the explicit detection of edges but uses image statistics (gradient distributions) instead, is presented by Fergus *et al.* (2006).

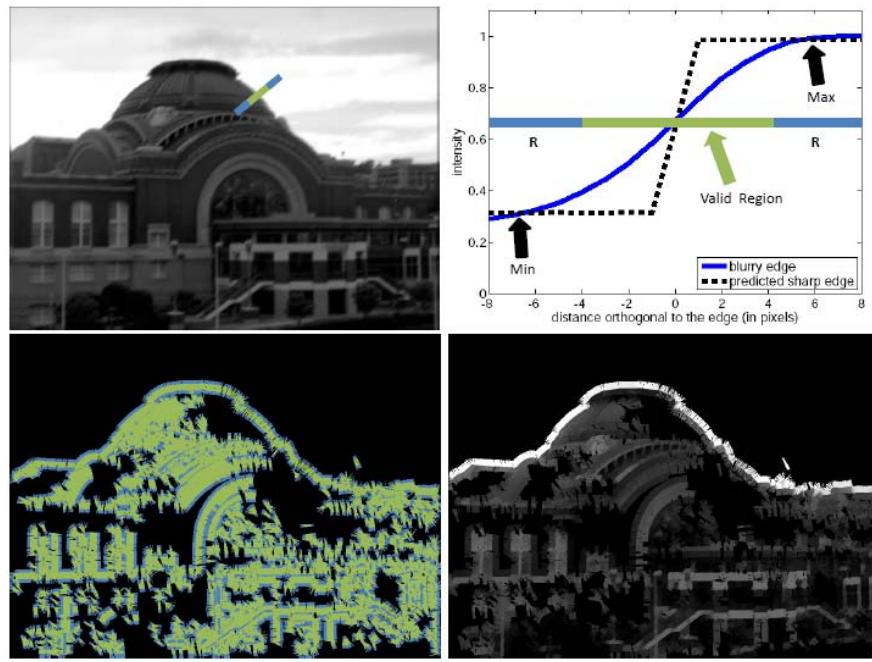


Figure 9.9: *Estimating the PSF without using a calibration pattern (Joshi et al. 2008).* (a) Input image with blue cross-section (profile) location. (b) Profile of sensed and predicted step edges. (c-d) Locations and values of the predicted colors near the edge locations.

[Note: Replace this with originals from LaTeX source directory and add (a–d) labels.]

9.2 High dynamic range imaging

As I mentioned earlier in this chapter, registered images taken at different exposures can be used to calibrate the radiometric response function of a camera. More importantly, they can help you create well-exposed photographs under challenging conditions, such as brightly lit scenes where any single exposure contains saturated (overexposed) and dark (underexposed) regions (Figure 9.10). This problem is quite common, because the natural world contains a range of radiance values that is far greater than can be captured with any photographic sensor or film (Figure 9.11). Taking a set of *bracketed exposures* (exposures selected by the camera in AEB mode to deliberately under- and over-expose the image) gives you the material from which to create a properly exposed photograph, as shown in Figure 9.12 (Reinhard et al. 2005, Freeman 2008).

While it is possible to combine pixels from different exposures directly into a final composite (Burt and Kolczynski 1993, Mertens et al. 2007), this approach runs the risk of creating contrast reversals and halos. Instead, the more common approach is to proceed in three stages:

1. Estimate the radiometric response function from the aligned images.
2. Estimate a *radiance map* by selecting and/or blending pixels from different exposures.

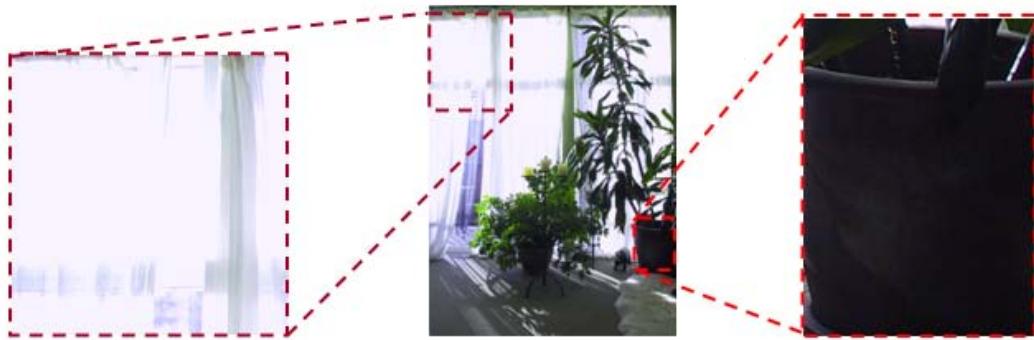


Figure 9.10: *Sample indoor image where the areas outside the window are overexposed and inside the room are too dark.*



Figure 9.11: *Relative brightness of different scenes, ranging from 1 inside a dark room lit by a monitor to 2,000,000 looking at the sun. Photos courtesy of Paul Debevec.
[Note: Get Paul's ok and ask for citation, if any]*



Figure 9.12: *A bracketed set of shots (using the camera's auto exposure bracket (AEB) mode) and the resulting high dynamic range (HDR) composite.*

3. Tone map the resulting high dynamic range (HDR) image back into a displayable gamut.

The idea behind estimating the radiometric response function is relatively straightforward (Mann and Picard 1995,Debevec and Malik 1997, Mitsunaga and Nayar 1999, Reinhard *et al.* 2005). Suppose you take three sets of images at different exposures (shutter speeds), say at ± 2 exposure values.⁷ If you knew the irradiance (exposure) E_i at each pixel (2.97), you could plot it versus the measured pixel value z_{ij} for each exposure time t_j , as shown in Figure 9.13.

Unfortunately, we do not know the irradiance values E_i , so these have to be estimated at the same time as the radiometric response function f , which can be written as (Debevec and Malik 1997)

$$z_{ij} = f(E_i t_j), \quad (9.2)$$

where t_j is the exposure time for the j th image. The inverse response curve f^{-1} is given by

$$f^{-1}(z_{ij}) = E_i t_j. \quad (9.3)$$

Taking logarithms of both sides (base 2 is convenient, as we can now measure quantities in EVs), we obtain

$$g(z_{ij}) = \log f^{-1}(z_{ij}) = \log E_i + \log t_j, \quad (9.4)$$

where $g = \log f^{-1}$, which maps pixel values z_{ij} into log irradiance, is the curve we will be estimating (Figure 9.13 turned on its side).

Debevec and Malik (1997) assume that the exposure times t_j are known. (Recall that these can be obtained from a camera's EXIF tags, but that they actually follow a power of 2 progression $\dots, 1/128, 1/64, 1/32, 1/16, 1/8, \dots$ instead of the marked $\dots, 1/125, 1/60, 1/30, 1/15, 1/8, \dots$ values—see Exercise 2.5.) The unknowns are therefore the per-pixel exposures E_i and the response values $g_k = g(k)$, where g can be discretized according to the 256 pixel values commonly observed in 8-bit images. (The response curves are calibrated separately for each color channel.)

In order to make the response curve smooth, Debevec and Malik (1997) add a second order smoothness constraint

$$\lambda \sum_k g''(k)^2 = \lambda \sum [g(k-1) - 2g(k) + g(k+1)]^2, \quad (9.5)$$

which is similar to the one used in snakes (4.35). Since pixel values are more reliable in the middle of their range (and the g function becomes singular near saturation values), they also add a weighting (hat) function $w(k)$ that decays to zero at both ends of the pixel value range,

$$w(z) = \begin{cases} z - z_{\min} & z \leq (z_{\min} + z_{\max})/2 \\ z_{\max} - z & z > (z_{\min} + z_{\max})/2. \end{cases} \quad (9.6)$$

⁷ Changing the shutter speed is preferable to changing the aperture, as the latter can modify the vignetting and focus. Using ± 2 “f-stops” (technically, exposure values, or EVs, since f-stops refer to apertures), is usually the right compromise between capturing a good dynamic range and having properly exposed pixels everywhere.

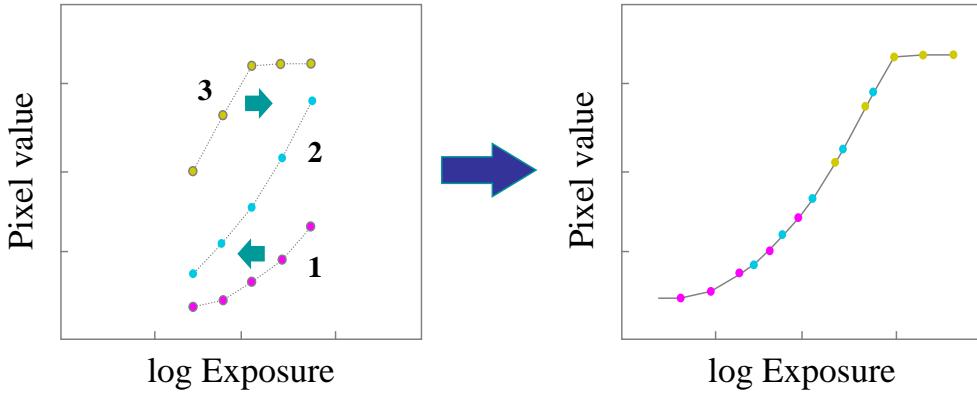


Figure 9.13: *Radiometric calibration using multiple exposures (Debevec and Malik 1997).* Corresponding pixel values are plotted as functions of log exposures (irradiance). The curves on the left are shifted to account for each pixel's unknown radiance until they all line up into a single smooth curve.

Putting all of these terms together, they obtain a least squares problem in the unknowns $\{g_k\}$ and $\{t_j\}$,

$$E = \sum_i \sum_j w(z_{i,j})[g(z_{i,j}) - \log E_i - \log t_j]^2 + \lambda \sum_k w(k)g''(k)^2. \quad (9.7)$$

(In order to remove the overall shift ambiguity in the response curve and irradiance values, the middle of the response curve is set to 0.) Debevec and Malik (1997) show how this can be implemented in 21 lines of Matlab code, which partially accounts for the popularity of their technique.

While Debevec and Malik (1997) assume that the exposure times t_j are known exactly, there is no reason why these additional variables cannot be thrown into the least squares problem, constraining their final estimated values to lie close to their nominal values \hat{t}_j with an extra term $\eta \sum_j (t_j - \hat{t}_j)^2$.

Figure 9.14 shows the recovered radiometric response function for a digital camera along with select (relative) radiance values in the overall radiance map. Figure 9.15 shows the bracketed input images captured on color film and the corresponding radiance map.

While Debevec and Malik (1997) use a general second-order smooth curve g to parameterize their response curve, Mann and Picard (1995) use a three-parameter

$$f(E) = \alpha + \beta E^\gamma \quad (9.8)$$

function, while Mitsunaga and Nayar (1999) use a low-order ($N \leq 10$) polynomial for the inverse response function g . Pal *et al.* (2004) derive a Bayesian model that estimates an independent smooth response function for each image, which can better model the more sophisticated (and hence less predictable) automatic contrast and tone adjustment performed in today's digital cameras.

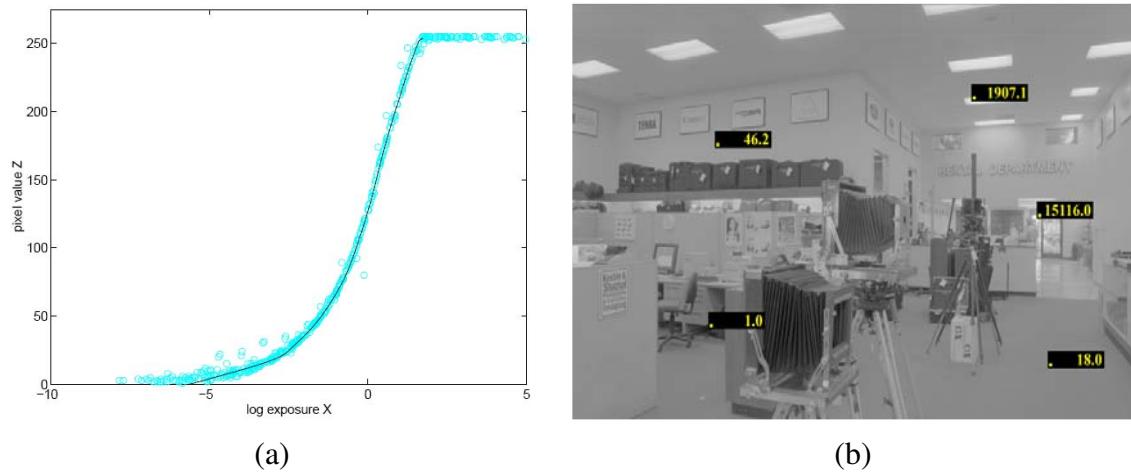


Figure 9.14: Recovered response function and radiance image for a real digital camera (DCS460) (Debevec and Malik 1997).

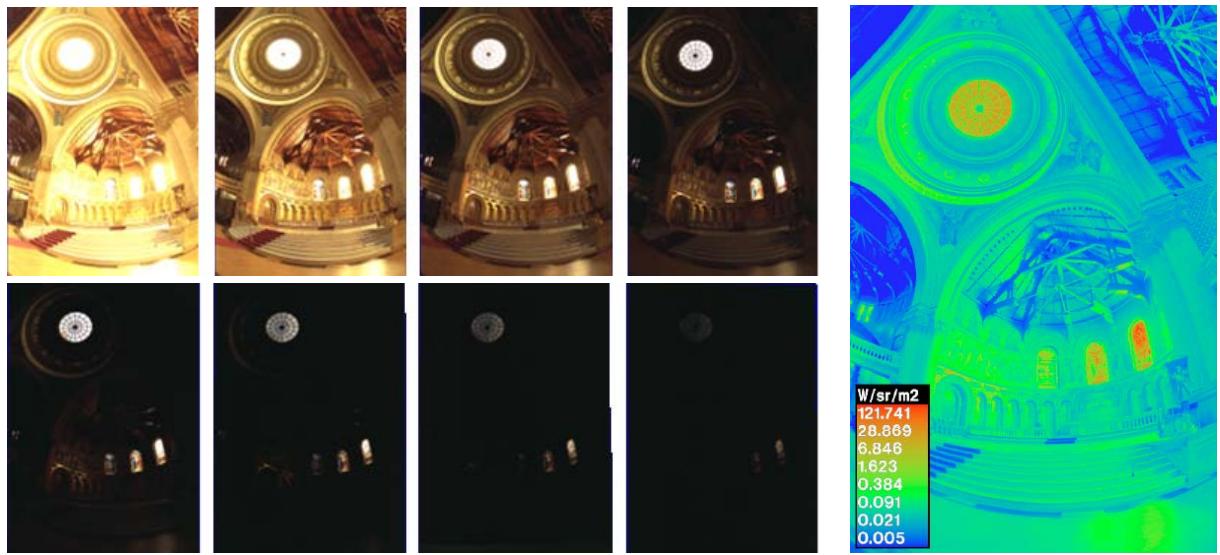


Figure 9.15: Bracketed set of exposures captured with a film camera and the resulting radiance image displayed in pseudocolor (Debevec and Malik 1997).

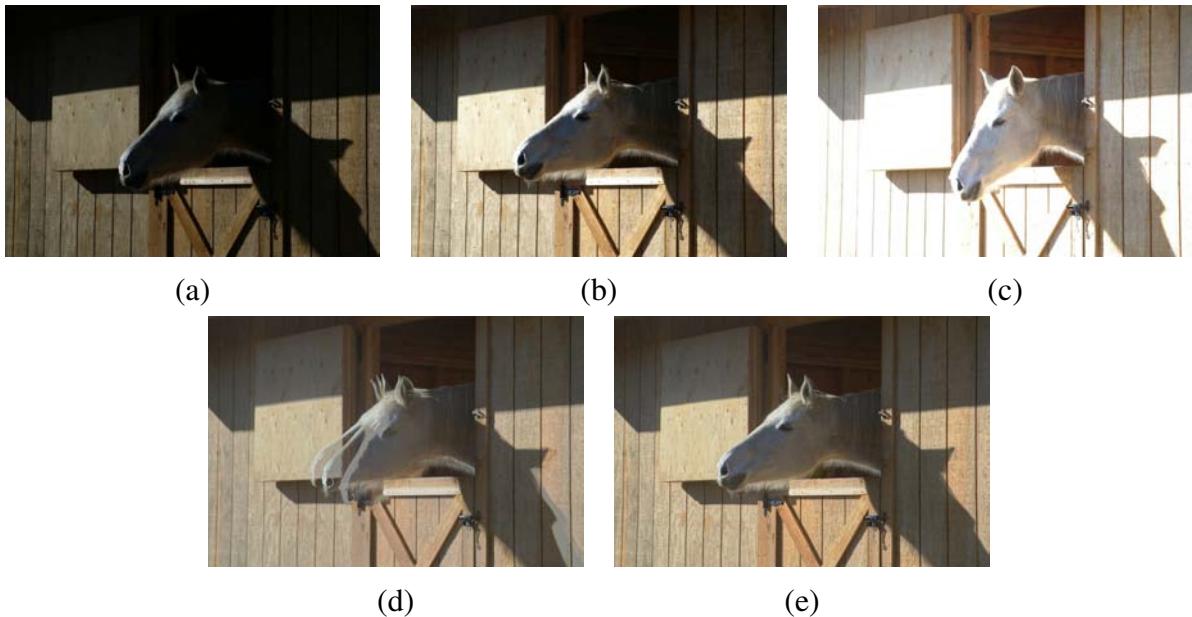


Figure 9.16: *Merging multiple exposures to create a high dynamic range composite* (Kang et al. 2003): (a–c) three different exposures; (d) merging the exposures using classic algorithms (note the ghosting due to the horse’s head movement); (e) merging the exposures with motion compensation.

Once the response function has been estimated, the second step in creating high dynamic range photographs is to merge the input images into a composite *radiance map*. If the response function and images were known exactly, i.e., they were noise free, you could use any non-saturated pixel value to estimate the corresponding radiance by mapping it through the inverse response curve $E = g(z)$.

Unfortunately, pixels are noisy, especially under low-light conditions when fewer photons arrive at the sensor. To compensate for this, Mann and Picard (1995) use the derivative of the response function as a weight in determining the final radiance estimate, since “flatter” regions of the curve tell us less about the incoming irradiance. Debevec and Malik (1997) use a the hat function (9.6) which accentuates mid-tone pixels while avoiding saturated values. Mitsunaga and Nayar (1999) show that in order to maximize the signal-to-noise ratio (SNR), the weighting function must emphasize both higher pixel values and larger gradients in the transfer function, i.e.,

$$w(z) = g(z)/g'(z), \quad (9.9)$$

where the weights w are used to form the final irradiance estimate

$$\log E_i = \frac{\sum_j w(z_{ij})[g(z_{ij}) - \log t_j]}{\sum_j w(z_{ij})}. \quad (9.10)$$

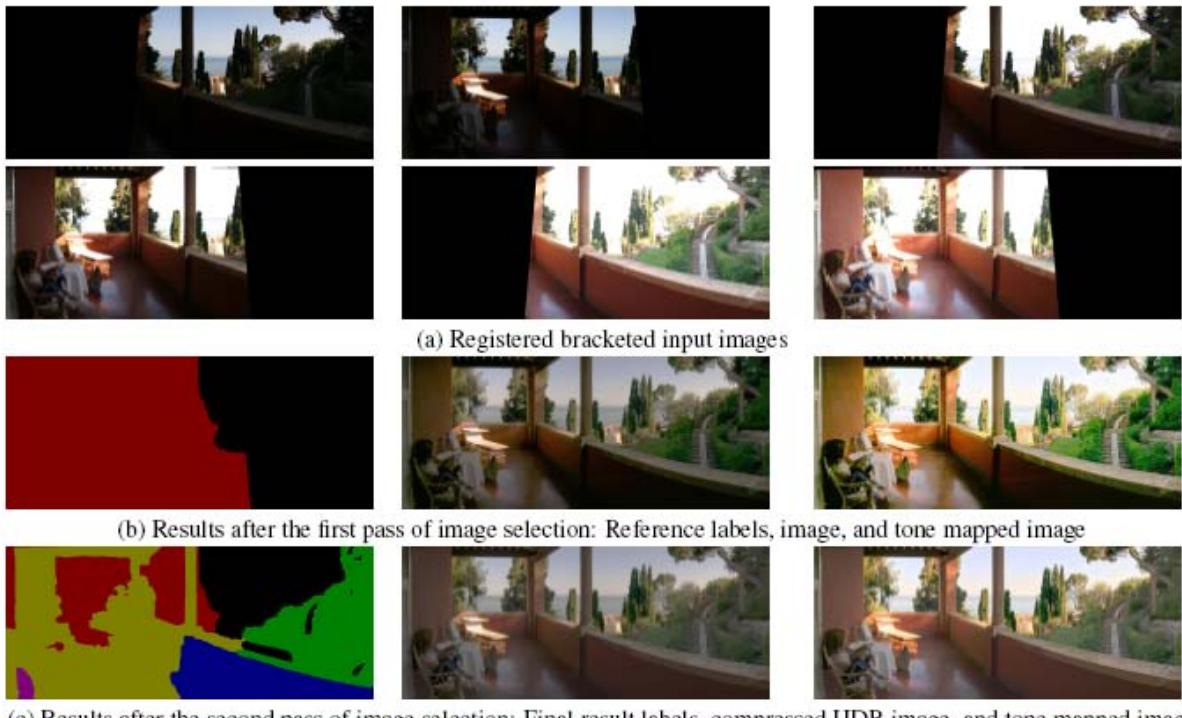


Figure 4. Results from a bracketed panoramic image sequence. The middle images of b and c are generated from the floating point radiance images but have a global compression curve applied to them instead of the local compression tone-mapping does.

Figure 9.17: *HDR merging with large amounts of motion* (Eden et al. 2006): [Note: Get the original photos from (Matt's?) LaTeX source directory and re-do the captions.]

Exercise 9.1 has you implement one of radiometric response function calibration techniques and then use it to create radiance maps.

Under real-world conditions, casually acquired images may not be perfectly registered and may contain moving objects. Ward (2003) uses a global (parametric) transform to align the input images, while Kang *et al.* (2003) present an algorithm that combines global registration with local motion estimation (optical flow) to accurately align the images before blending their radiance estimates (Figure 9.16). Since the images may have widely different exposures, care must be taken when producing the motion estimates, which must themselves be checked for consistency to avoid the creation of ghosts and object fragments.

Even this approach, however, may not work when the camera is simultaneously undergoing large panning motions and exposure changes, which is a common occurrence in casually acquired panoramas. Under such conditions, different parts of the image may be seen at one or more exposures. Devising a method to blend all of these different sources while avoiding sharp transitions and dealing with scene motion is a challenging problem. One approach is to first find a consensus

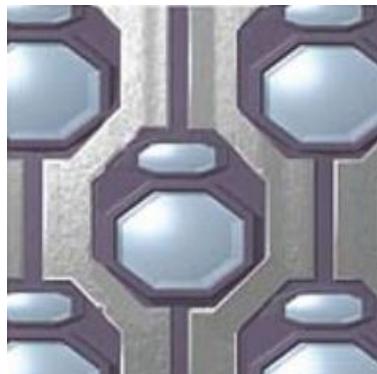


Figure 9.18: *Fuji SuperCCD high dynamic range image sensor. The paired large and small active areas provide two different effective exposures.*

mosaic and to then selectively compute radiances in under- and over-exposed regions (Eden *et al.* 2006), as shown in Figure 9.17.

In the long term, the need to compute high dynamic range images from multiple exposures may be eliminated by advances in camera sensor technology (Figure 9.18) (Yang *et al.* 1999, Nayar and Mitsunaga 2000, Nayar and Branzoi 2003, Kang *et al.* 2003, Narasimhan and Nayar 2005, Tumblin *et al.* 2005). However, the need to blend such images and to tone map them to a pleasing final result will likely remain.

HDR image formats

Before I discuss techniques for mapping HDR images back to a displayable gamut, I should mention the commonly used formats for storing HDR images.

If storage space is not an issue, storing each of the R, G, and B values as a 32-bit IEEE float is the best (and simplest) solution. The commonly used Portable PixMap (.ppm) format, which supports both uncompressed ASCII and raw binary encodings of values, can be extended to a Portable FloatMap (.pfm) format by modifying the header. TIFF also supports full floating point values.

A more compact representation is the Radiance format (.pic, .hdr) (Ward 1994), which uses a single common exponent and per-channel mantissas (9.19b). An intermediate encoding, OpenEXR from ILM,⁸ uses 16-bit floats for each channel (9.19c), which is a format supported natively on most modern GPUs. Ward (2004) describes these (and other) data formats such as LogLuv (Larson 1998) in more detail, as do the books by (Reinhard *et al.* 2005) and (Freeman 2008). An even more recent HDR image format is Microsoft's HDPhoto, which is under consideration as a JPEG XR standard.

⁸ <http://www.openexr.net/>

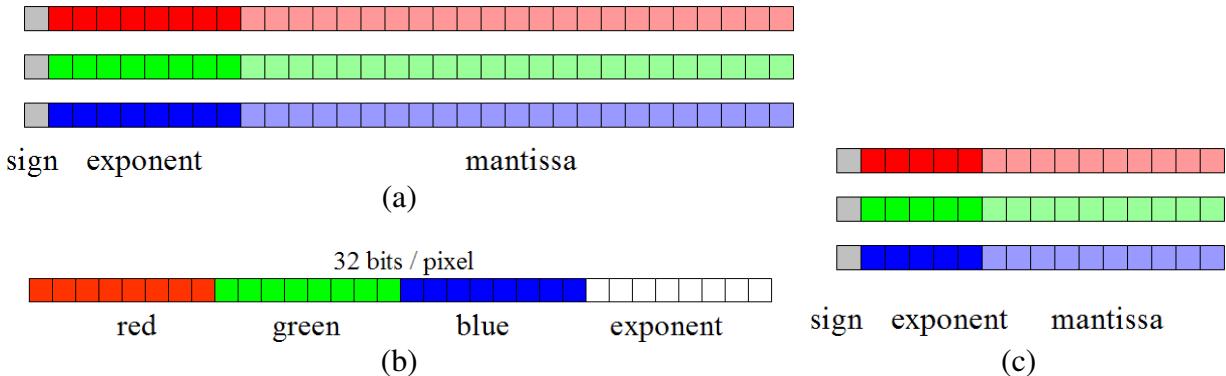


Figure 9.19: *HDR image encoding formats*: (a) Portable PixMap (.ppm); (b) Radiance (.pic, .hdr); (c) OpenEXR (.exr).

[Note: Lots of software and HDR image pointers in Fredo's 06_Bila_HDR.pdf (p. 116) and 05_HDR.pdf:]

<http://www.luminous-landscape.com/tutorials/hdr.shtml>
<http://www.anyhere.com/gward/hdrenc/>
<http://www.debevec.org/IBL2001/NOTES/42-gward-cic98.pdf>
<http://www.openexr.com/>
<http://gl.ict.usc.edu/HDRShop/>
http://www.dpreview.com/learn/?/Glossary/Digital_Imaging/Dynamic_Range_01.htm
http://www.normankoren.com/digital_tonality.html
<http://www.anyhere.com/>

9.2.1 Tone mapping

Once a radiance map has been computed, it is usually necessary to display it on a lower gamut (i.e., 8-bit) screen or printer. A variety of *tone mapping* techniques have been developed for this purpose, which involve either computing spatially varying transfer functions or reducing image gradients to fit the the available dynamic range (Reinhard *et al.* 2005). [Note: I have about 20 references to different algorithms. Should I list them here, or just refer people to the (Reinhard *et al.* 2005) book and Frédo's course notes?]

The simplest way to compress a high dynamic range radiance image into a low-dynamic range gamut is to use a global transfer curve (Larson *et al.* 1997). Figure 9.20 shows one such example, where a gamma curve is used to map an HDR image back into a displayable gamut. If gamma is applied separately to each channel (Figure 9.20b), the colors become muted (less saturated), since higher-valued color channels contribute less (proportionately) to the final color. Splitting the image



Figure 9.20: *Global tone mapping*: (a) input HDR image, linearly mapped; (b) gamma applied to each color channel independently; (c) gamma applied to intensity (colors are less washed out). Images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.

up into its luminance and chrominance (say $L^*a^*b^*$) components §2.3.2, applying the global mapping to the luminance channel, and then reconstituting a color image works better (Figure 9.20c).

Unfortunately, when the image has a really wide range of exposures, this global approach still fails to preserve details in regions with widely varying exposures. What is needed, instead, is something akin to the dodging and burning performed by photographers in the darkroom. Mathematically, this is similar to dividing each pixel by the *average* brightness in a region around that pixel.

Figure 9.21 shows how this process works. As before, the image is split into its luminance and chrominance channels. The log luminance image

$$H(x, y) = \log L(x, y) \quad (9.11)$$

is then low-pass filtered to produce a *base layer*

$$H_L(x, y) = B(x, y) * H(x, y), \quad (9.12)$$

and a high-pass *detail layer*

$$H_H(x, y) = H(x, y) - H_L(x, y). \quad (9.13)$$

The base layer is then contrast reduced by scaling to the desired log-luminance range,

$$H'_H(x, y) = s H_H(x, y). \quad (9.14)$$

and added to the detail layer to produce the new log-luminance image

$$I(x, y) = H'_H(x, y) + H_L(x, y), \quad (9.15)$$



(a)

(b)

Figure 9.21: *Local tone mapping using linear filters:* (a) low-pass and high-pass filtered log luminance images and color (chrominance) image; (b) resulting tone-mapped image (after attenuating the low-pass log luminance image) shows visible halos around the trees. Images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.

which can then be exponentiated to produce the tone-mapped (compressed) luminance image. Note that this process is equivalent to dividing each luminance value by (a monotonic mapping of) the average log-luminance value in a region around that pixel.

Figure 9.21 shows the low-pass and high-pass log luminance image and the resulting tone-mapped color image. Note how the detail layer has visible *halos* around the high-contrast edges, which are visible in the final tone-mapped image. This is because linear filtering, which is not edge preserving, produces halos in the detail layer (Figure 9.22).

The solution to this problem is to use an edge-preserving filter to create the base layer. Durand and Dorsey (2002) study a number of such edge-preserving filters, including anisotropic and robust anisotropic diffusion, and select bilateral filtering §3.2.2 as their edge-preserving filter. (See (Farbman *et al.* 2008) for a more recent paper that argues in favor of using a weighted least squares (WLF) filter as an alternative to the bilateral filter.) Figure 9.23 shows how replacing the linear low-pass filter with a bilateral filter produces tone mapped images with no visible halos. Figure 9.24 summarizes the complete information flow in this process, starting with the decomposition into log luminance and chrominance images, bilateral filtering, contrast reduction, and re-composition into the final output image.

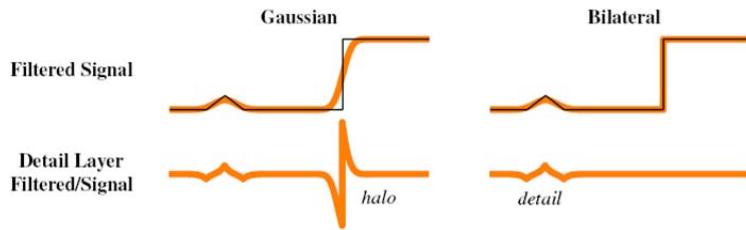


Figure 9.22: *Gaussian vs. bilateral filtering* (Petschnigg et al. 2004): A Gaussian low-pass filter blurs across all edges and therefore creates strong peaks and valleys in the detail image that cause halos. The bilateral filter does not smooth across strong edges and thereby reduces halos while still capturing detail.



Figure 9.23: *Local tone mapping using bilateral filter* (Durand and Dorsey 2002): (a) low-pass and high-pass bilateral filtered log luminance images and color (chrominance) image; (b) resulting tone-mapped image (after attenuating the low-pass log luminance image) shows no halos. Images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.

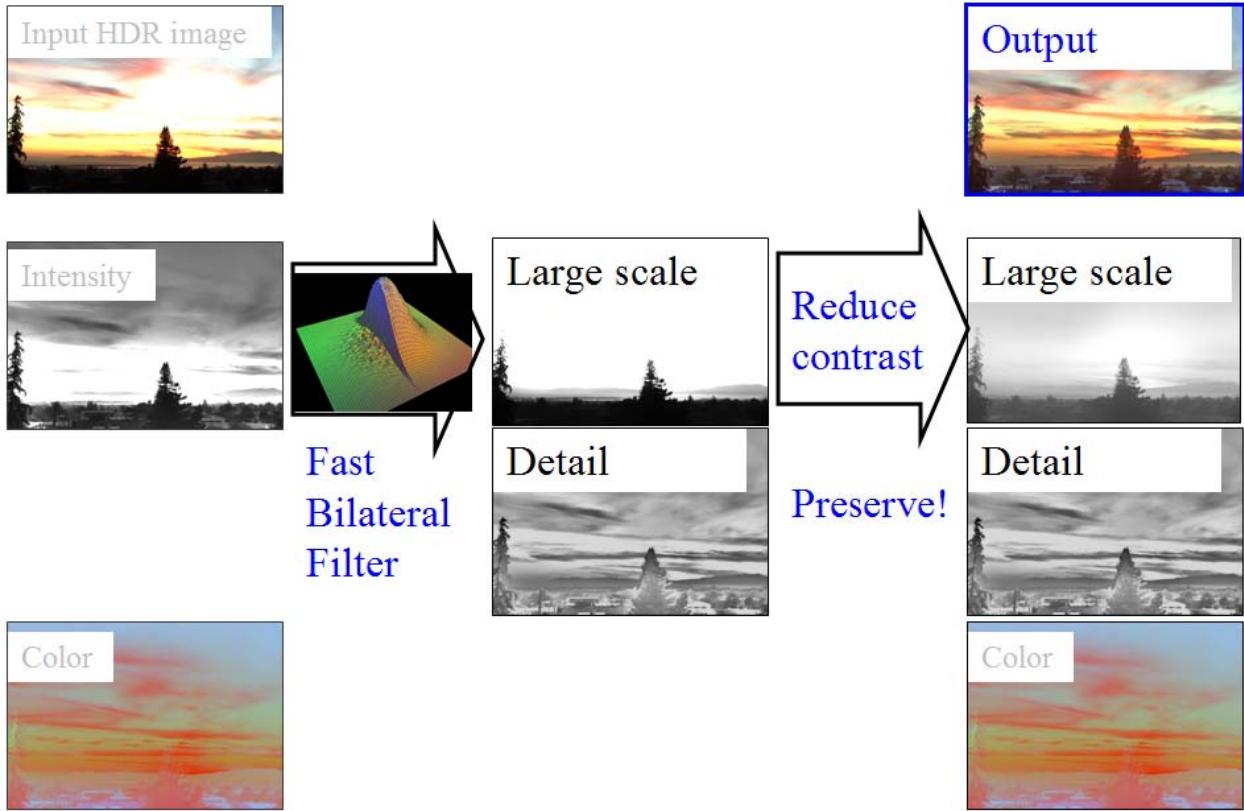


Figure 9.24: *Local tone mapping using bilateral filter (Durand and Dorsey 2002): summary of algorithm workflow. Images courtesy of Frédo Durand, MIT 6.815/6.865 course on Computational Photography.*

An alternative to compressing the base layer is to compress its *derivatives*, i.e., the gradient of the log-luminance image (Fattal *et al.* 2002). Figure 9.25 illustrates this process. The log-luminance image is differentiated to obtain a gradient image

$$H'(x, y) = \nabla H(x, y). \quad (9.16)$$

This gradient image is then attenuated by a spatially varying attenuation function $\Phi(x, y)$,

$$G(x, y) = H'(x, y) \Phi(x, y), \quad (9.17)$$

The attenuation function $I(x, y)$ is designed to attenuate large scale brightness changes (Figure 9.26a) and is designed to take into account gradients at different spatial scales (Fattal *et al.* 2002).

After attenuation, the resulting gradient field is re-integrated by solving a first-order variational (least squares) problem,

$$\min \int \int \|\nabla I(x, y) - G(x, y)\|^2 dx dy \quad (9.18)$$

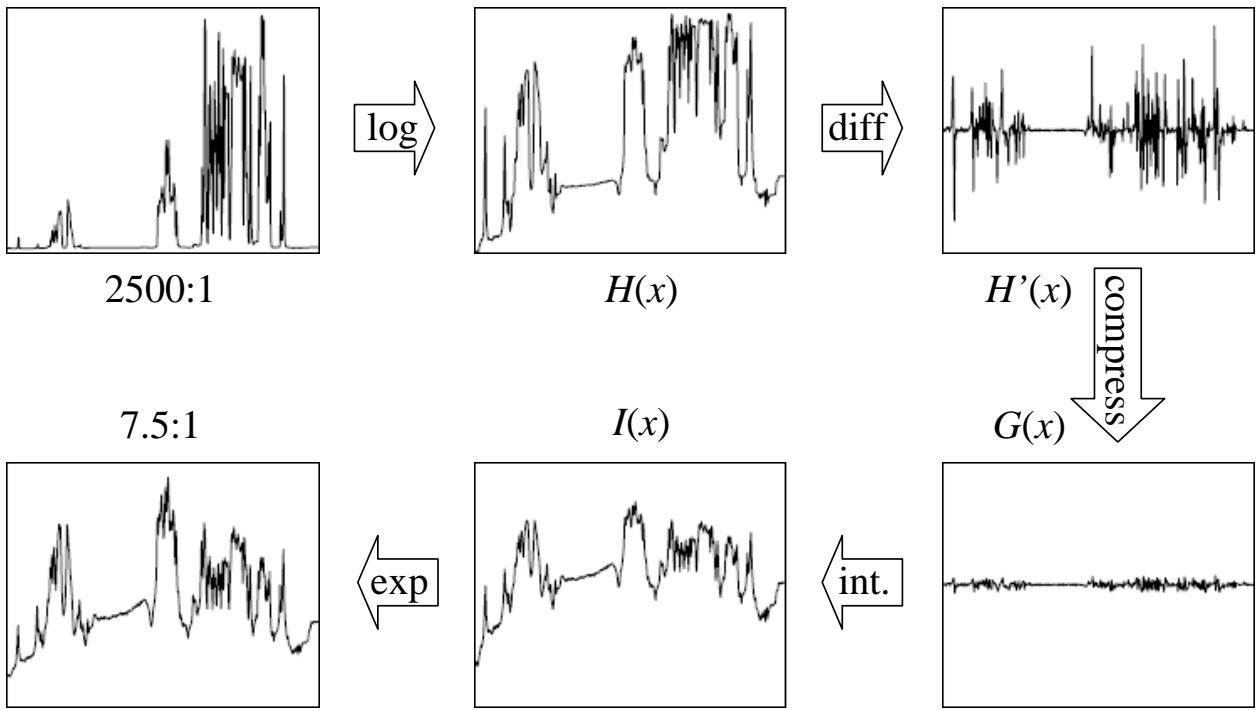


Figure 9.25: *Gradient domain tone mapping* (Fattal et al. 2002). The original image with a dynamic range of 2415:1 is first converted into the log domain, $H(x)$, and its gradients are computed, $H'(x)$. These are attenuated (compressed) based on local contrast, $G(x)$, and integrated to produce the new logarithmic exposure image $I(x)$, which is exponentiated to produce the final intensity image, whose dynamic range is 7.5:1.

to obtain the compressed log-luminance image $I(x, y)$. This least squares problem is the same that was used for Poisson blending §8.3.3 and was first introduced in our study of regularization §3.6.1 (3.99). It can efficiently be solved using techniques such as multi-grid and hierarchical basis preconditioning (Fattal et al. 2002, Szeliski 2006b, Farbman et al. 2008). Once the new luminance image has been computed, it is combined with the original color image using

$$C_{\text{out}} = \left(\frac{C_{\text{in}}}{L_{\text{in}}} \right)^2 L_{\text{out}}, \quad (9.19)$$

where $C = (R, G, B)$ and L_{in} and L_{out} are the original and compressed luminance images. The exponent s controls the saturation of the colors and is typically in the range $s \in [0.4, 0.6]$. Figure 9.26b shows the final tone-mapped color image, which shows no visible halos despite the extremely large variation in input radiance values.

Yet another alternative to these two approaches is to perform the local dodging and burning using a locally scale-selective operator (Reinhard et al. 2002). Figure 9.27 shows how such a scale selection operator can determine a radius (scale) that only includes similar color values within the



Figure 9.26: *Gradient domain tone mapping* (*Fattal et al. 2002*): (a) attenuation map, with darker values corresponding to more attenuation; (b) final tone-mapped image.

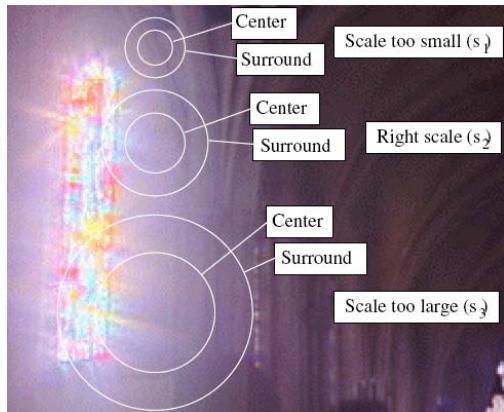


Figure 9.27: *Scale selection for tone mapping* (*Reinhard et al. 2002*).

inner circle while avoiding much brighter values in the surrounding circle. (In practice, a difference of Gaussians normalized by the inner Gaussian response is evaluated over a range of scales, and the largest scale whose metric is below a threshold is selected (*Reinhard et al. 2002*)).

What all of these techniques have in common is that they adaptively attenuate or brighten different regions of the image so that they can be displayed in a limited gamut without loss of contrast. *Lischinski et al. (2006b)* introduce an *interactive* technique that performs this operation by interpolating a set of sparse user-drawn adjustments (strokes and associated exposure value corrections) to a piecewise-continuous exposure correction map (Figure 9.28). The interpolation is performed by minimizing a locally weighted least square (WLS) variational problem,

$$\min \int \int w_d(x, y) \|f(x, y) - g(x, y)\|^2 dx dy + \lambda \int \int w_s(x, y) \|\nabla f(x, y)\|^2 dx dy, \quad (9.20)$$



Figure 9.28: *Interactive local tone mapping* (Lischinski et al. 2006b): (a) user drawn strokes with associated exposure values $g(x, y)$ (b) corresponding piecewise smooth exposure adjustment map $f(x, y)$.

where $g(x, y)$ and $f(x, y)$ are the input and output log exposure (attenuation) maps (Figure 9.28). The data weighting term $w_d(x, y)$ is 1 at stroke locations and 0 elsewhere. The smoothness weighting term $w_s(x, y)$ is inversely proportional to the log-luminance gradient,

$$w_s = \frac{1}{\|\nabla H\|^\alpha + \epsilon} \quad (9.21)$$

and hence encourages the $f(x, y)$ map to be smoother in low-gradient areas than along high-gradient discontinuities.⁹ The same approach can also be used for fully automated tone mapping by setting target exposure values at each pixel and allowing the weighted least squares to convert these into piecewise smooth adjustment maps.

The weighted least squares algorithm, which was originally developed for image colorization applications (Levin et al. 2004), has recently been applied for general edge-preserving smoothing in applications such as contrast enhancement (Bae et al. 2006) where the bilateral filtering was previously used (Farbman et al. 2008). [Note: Should I discuss (Bae et al. 2006) somewhere in this chapter? Should I give more details about (Farbman et al. 2008)?]

Given the wide range of locally adaptive tone mapping algorithms that have been developed, which ones should be used in practice? Freeman (2008) has a great discussion of commercially available algorithms, their artifacts, and the parameters that can be used to control them. He also has a wealth of tips for HDR photography and workflow. I highly recommend this book for

⁹ In practice, the x and y discrete derivatives are weighted separately (Lischinski et al. 2006b). Their default parameter settings are $\lambda = 0.2$, $\alpha = 1$, and $\epsilon = 0.0001$.

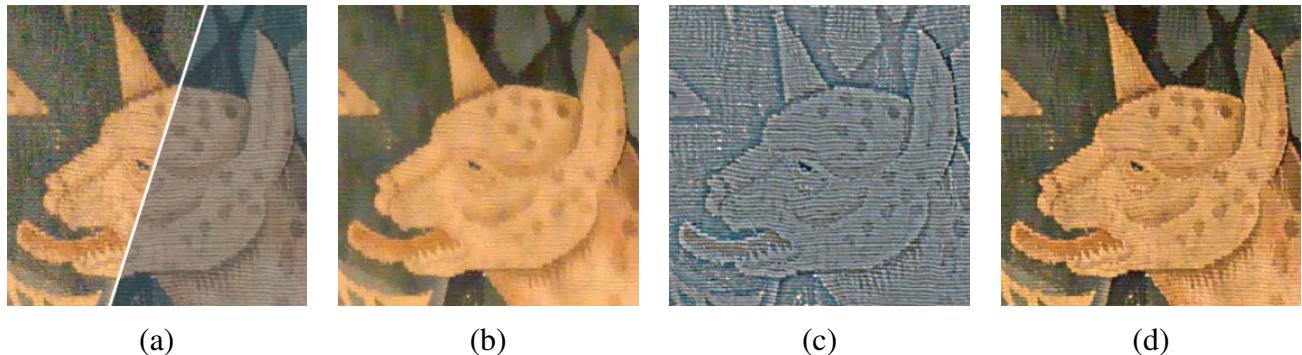


Figure 9.29: *Detail transfer in flash/no-flash photography* (Petschnigg et al. 2004): (a) details of input ambient and flash images A and F ; (b) joint bilaterally filtered no-flash image A^{NR} ; (c) detail layer F^{Detail} computed from the flash image F ; (e) final merged image A^{Final} .

anyone contemplating doing additional research (or personal photography) in this area.

[Note: Discuss perceptual metrics (suggestion by Alyosha)?]

[Note: Be sure to look at Fredo's slides for more details on both tone mapping (06_Bila_HDR.ppt) and other computational photography papers (10_Gradient_6.ppt). Also, check out HDRShop, <http://www.debevec.org/HDRShop>.]

9.2.2 Application: Flash photography

While high dynamic range imaging combines images of a scene taken at different exposures, it is also possible to combine flash and non-flash images together to achieve better exposure, color balance, and to reduce noise (Eisemann and Durand 2004, Petschnigg et al. 2004).

The problem with flash images is that their color is often unnatural (fails to capture the ambient illumination), there may be strong shadows or specularities, and there is a radial falloff in brightness away from the camera (Figures 9.1b and 9.29a). Non-flash photos taken under low light conditions often suffer from excessive noise (because of the high ISO gains and low photon counts) and blur (due to longer exposures). Wouldn't it be wonderful if a non-flash photo taken just before the flash goes off could be combined with the flash photo to produce an image with good color values, sharpness, and low noise?¹⁰ [Note: The FujiFilm FinePix F40fd camera takes a pair of flash and no flash images, and then it lets you select which one to keep. See <http://technabob.com/blog/2007/01/22/fuji-f40fd-flash-comparison-and-face-recognition-camera/>, http://www.bookofjoe.com/2007/03/flashno_flash_c.html, <http://www.hammacher.com/publish/73698.asp>. The dpreview site, <http://www.dpreview.com/reviews/specs/>

¹⁰ In fact, the now discontinued FujiFilm FinePix F40fd camera takes a pair of flash and no flash images in quick successions, but then it only lets you decide which one to keep.

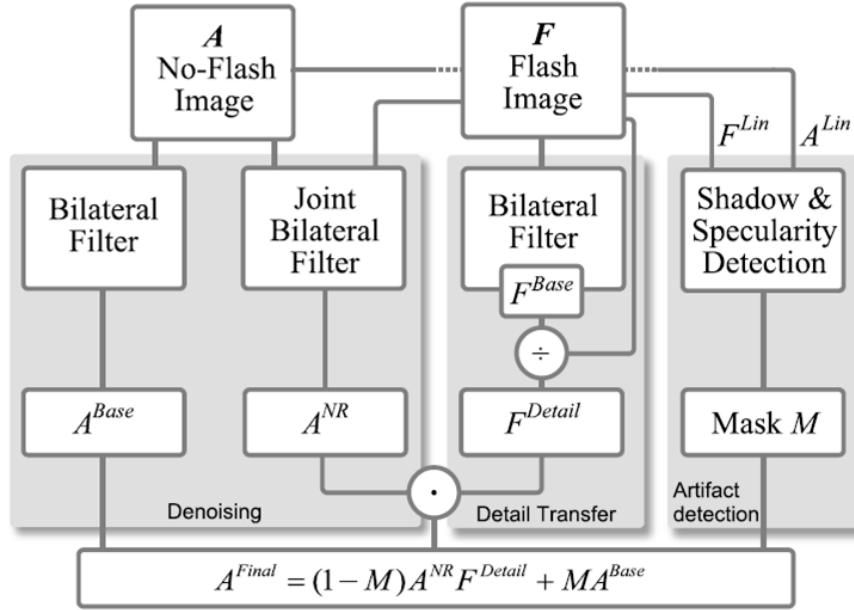


Figure 9.30: *Flash/no-flash photography algorithm* (Petschnigg et al. 2004). The ambient (no-flash) image A is filtered with a regular bilateral filter to produce A^{Base} , which is used in shadow and specularity regions, and a joint bilterally filtered noise reduced image A^{NR} . The flash image F is bilaterally filtered to produce a base image F^{Base} and a detail (ratio) image F^{Detail} , which is used to modulate the de-noised ambient image. The shadow/specularity mask M is computed by comparing linearized versions of the flash and no-flash images.

[FujiFilm/fuji_finepixf40fd.asp](#) doesn't say anything about this.]

Petschnigg et al. (2004) approach this problem by first filtering the no-flash (ambient) image A with a variant of the bilateral filter called the *joint bilateral filter*¹¹ in which the range kernel (3.36)

$$r(i, j, k, l) = \exp\left(-\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right) \quad (9.22)$$

is evaluated on the flash image F instead of the ambient image A , since the flash image is less noisy and hence has more reliable edges. (Figure 9.29b). Because the contents of the flash image can be unreliable inside and at the boundaries of shadows and specularities, these are detected and the regular bilateral image A^{Base} is used instead (Figure 9.30).

The second stage of their algorithm computes a flash detail image

$$F^{Detail} = \frac{F + \epsilon}{F^{Base} + \epsilon}, \quad (9.23)$$

¹¹ Eisemann and Durand (2004) call this the *cross bilateral filter*.

where F^{Base} is a bilaterally filtered version of the flash image F and $\epsilon = 0.02$. This detail image (Figure 9.29c) encodes details that may have been filtered away from the noise-reduced no-flash image A^{NR} , as well as additional details created by the flash camera, which often add crispness. The detail image is used to modulate the noise-reduced ambient image A^{NR} to produce the final results

$$A^{Final} = (1 - M)A^{NR}F^{Detail} + MA^{Base} \quad (9.24)$$

shown in Figures 9.1b and 9.29d.

[Eisemann and Durand \(2004\)](#) present an alternative algorithm that shares some of the same basic concepts. Both papers are well worth reading and contrasting (Exercise 9.6).

Flash images can also be used for a variety of additional applications such as extracting more reliable foreground mattes of objects ([Raskar et al. 2004](#), [Sun et al. 2006](#)). Flash photography is just one instance of the more general topic of *active illumination*, which is discussed in more detail in ([Raskar and Tumblin 2009](#)).

9.3 Super-resolution and blur removal

classic ([Keren et al. 1988](#), [Cheeseman et al. 1993](#), [Irani and Peleg 1991](#), [Mann and Picard 1994](#), [Chiang and Boult 1996](#), [Bascle et al. 1996](#), [Capel and Zisserman 1998](#), [Smelyanskiy et al. 2000](#), [Capel and Zisserman 2000](#)) [Note: [Tekalp's book](#) also has a section: check it out] [Note: See reviewers of Wang PAMI submission]

NIPS 2006 paper 501

See Exercise 9.8 for some commonly used technique, which should be mentioned here.

See also Capel's thesis <http://www.robots.ox.ac.uk/~vgg/publications/html/index.html>

non-linear (learned) priors ([Freeman et al. 2000](#), [Freeman et al. 2002](#), [Baker and Kanade 2002](#), [Capel and Zisserman 2001](#)) (and check references therein)

close relationship to texture synthesis ([Efros and Leung 1999](#)) and image analogies ([Efros and Freeman 2001](#), [Hertzmann et al. 2001](#)). [Note: Newer variant on image analogies is ([Cheng et al. 2008](#)).]

where to do foreshortening?

blur removal, closely related to de-noising; see ([Joshi et al. 2008](#)) as a starting point

Latest work from MSR Asia (multi-image and coarse-to-fine) ([Yuan et al. 2007](#), [Yuan et al. 2008](#)). [Yuan et al. \(2008\)](#) have a nice literature review, including some survey papers I should read and cite.

Removing Camera Shake from a Single Photograph ([Fergus et al. 2006](#)) and more recent work by Anat Levin

also, coded aperture techniques ([Levin et al. 2007](#))

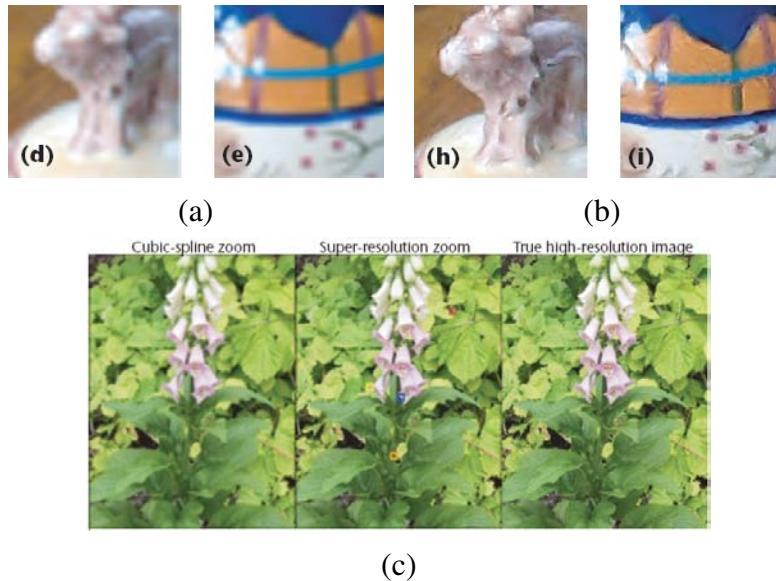


Figure 9.31: *Example-based super-resolution (Freeman et al. 2002): close-ups of (a) cubic interpolation and (b) example-based super-resolution; (c) comparison on full-sized image.*

9.3.1 Color image de-mosaicing

From Bayer patterns...

[Note: Re-use some material from (Bennett et al. 2006).]

Here's a survey from 2002 (recommended by Bill Freeman), (Longere et al. 2002), along with more recent work by Bill and his students, (Tappen et al. 2003).

Frédo's course notes talk about median filtering the color difference image after naive interpolation, but don't give a citation.

Look at this as an irregular super-resolution problem.



Figure 9.32: *Challenging test image for color filter array de-mosaicing: strong iso-luminant color edges are challenging for most algorithms.*

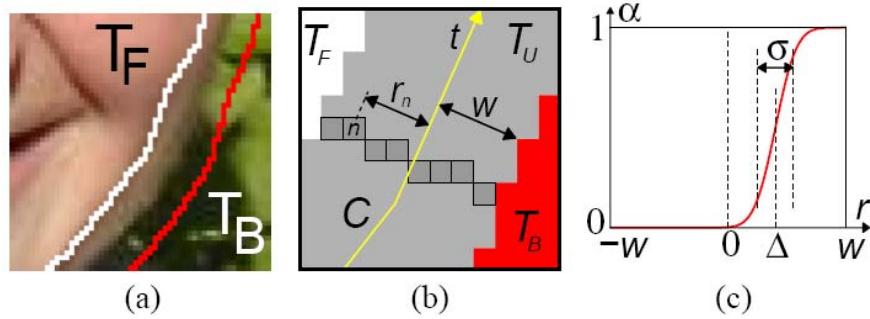


Figure 9.33: *Softening a hard segmentation boundary (border matting)* (Rother et al. 2004): (a) the region surrounding a segmentation boundary where pixels of mixed foreground and background colors are visible; (b) pixel values along the boundary are used to compute a soft alpha matte; (c) at each point along the curve t , a displacement Δ and a width σ are estimated.

Not sure if these are relevant: (Zomet and Peleg 2002, Omer and Werman 2004)

9.4 Image matting and compositing

Image matting and compositing is the process of cutting a foreground object out of one image and pasting it against a novel background (Smith and Blinn 1996, Wang and Cohen 2007a). It is commonly used in television and film production to composite a live actor in front of computer-generated imagery such as weather maps or 3D virtual characters and scenery (Wright 2006, Brinkmann 2008).

We have already seen a number of tools for interactively segmenting objects in an image, including snakes §4.4.1, scissors §4.4.2, and Grab Cut segmentation §4.5.4. While these techniques can generate reasonable pixel-accurate segmentations, they fail to capture the subtle interplay of foreground and background colors at *mixed pixels* along the boundary (Szeliski and Golland 1999) (Figure 9.33a).

In order to successfully copy a foreground object from one image to another one without visible discretization artifacts, we need to *pull a matte*, i.e., to estimate a soft opacity channel α and the uncontaminated foreground colors F from the input composite image C . Recall from §3.1.3 (Figure 3.4), that the compositing equation (3.8) can be written as

$$C = (1 - \alpha)B + \alpha F. \quad (9.25)$$

This operator attenuates the influence of the background image B by a factor $(1 - \alpha)$ and then adds in the (partial) color values corresponding to the foreground element F .

While the compositing operation is easy to implement, the reverse *matting* operation of estimating F , α , and B given an input image C is much more challenging (Figure 9.34). To see

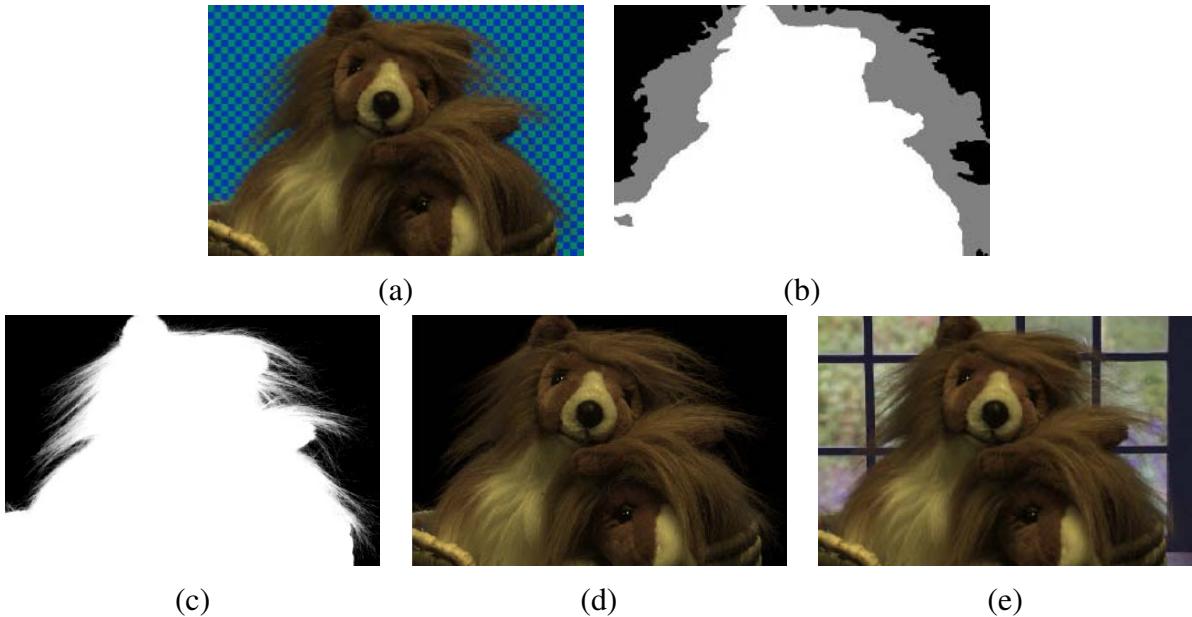


Figure 9.34: *Natural image matting* ([Chuang et al. 2001](#)): (a) input image with a “natural” (non-constant) background; (b) hand-drawn trimap—gray indicates unknown regions; (c) extracted alpha map; (d) extracted (premultiplied) foreground colors; (e) composite over a novel background.

why, observe that while the composite pixel color C provides 3 measurements, the F , α , and B unknowns have a total of 7 degrees of freedom. Devising techniques to estimate these unknowns despite the underconstrained nature of the problem is the essence of image matting.

In this section, I review a number of image matting techniques. I begin with *blue screen matting*, which assumes that the background is a constant known color, and discuss its variants, namely two-screen matting (when multiple backgrounds can be used) and difference matting, where the known background is arbitrary. I then discuss local variants of *natural image matting*, where both the foreground and background are unknown. In these applications, it is usual to designate a *trimap*, i.e., a 3-way labelling of the image into foreground, background, and unknown regions (Figure 9.34b). Next, I present some global optimization approaches to natural image matting. Finally, I discuss variants on the matting problem, including shadow matting, flash matting, and environment matting.

9.4.1 Blue screen matting

Blue screen matting involves filming an actor (or object) in front of a constant colored background. While originally bright blue was the preferred color, bright green is now more commonly used ([Wright 2006](#), [Brinkmann 2008](#)). [Smith and Blinn \(1996\)](#) discuss a number of techniques for blue

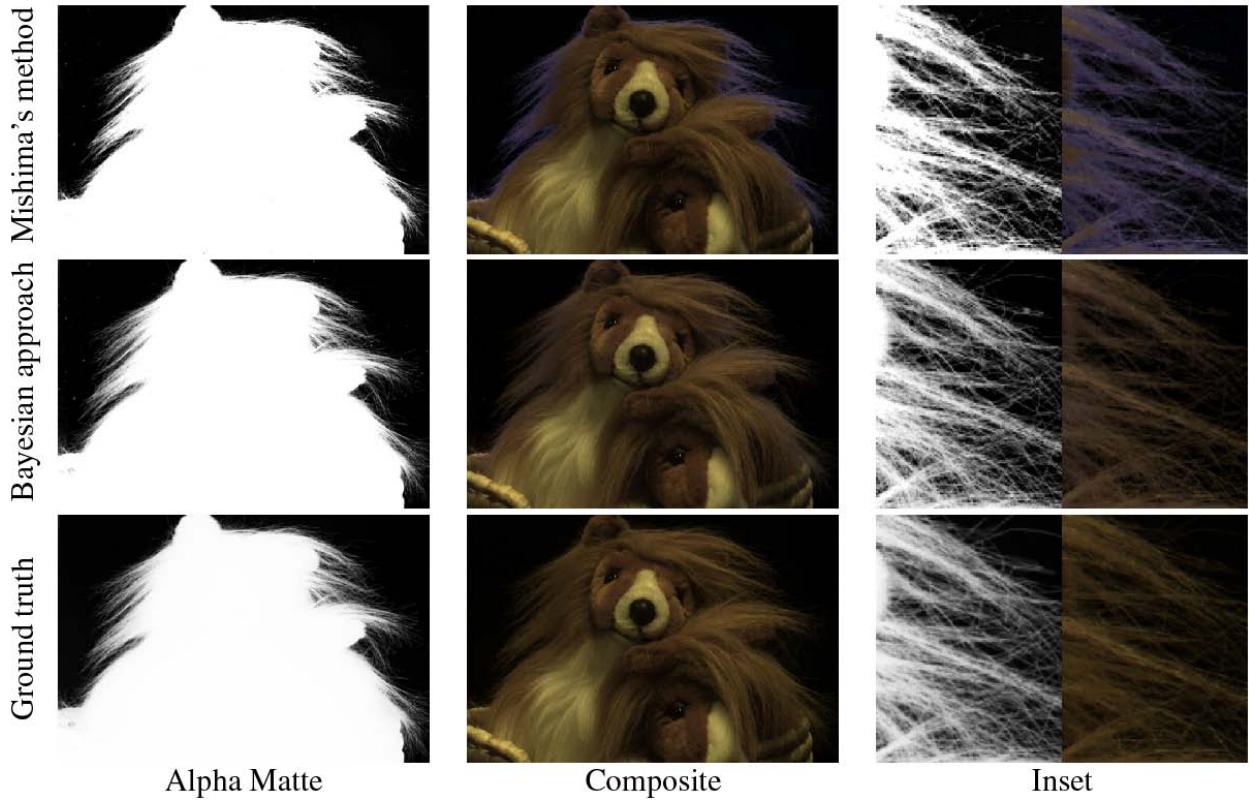


Figure 9.35: *Blue-screen matting results (Chuang et al. 2001)*. Mishima’s method produces visible blue spill (color fringing in the hair), while Chuang’s method produces accurate results.

screen matting, which are mostly described in patents rather than the open research literature. Early techniques used linear combination of object color channels along with user-tuned parameters to estimate the opacity α .

Chuang *et al.* (2001) describe a newer technique called Mishima’s algorithm, which involves fitting two polyhedral surfaces (centered at the mean background color) separating the foreground and background color distributions and then measuring the relative distance of a novel color to these surfaces to estimate α (Figure 9.36e). While this technique works well in many studio settings, it can still suffer from *blue spill*, where translucent pixels around the edges of an object acquire some of the background blue coloration (Figure 9.35).

Two screen matting

In their paper, Smith and Blinn (1996) also introduce an algorithm called *triangulation matting* that uses more than one known background color to over-constrain the equations required to estimate the opacity α and foreground color F .

For example, consider in the compositing equation (9.25) setting the background color to black, i.e., $B = 0$. The resulting composite image C is therefore equal to αF . Replacing the background color with a different known non-zero value B now results in

$$C - \alpha F = (1 - \alpha)B, \quad (9.26)$$

which is an overconstrained set of (color) equations for estimating α . In practice, B should be chosen so as not to saturate C , and for best accuracy, several values of B should be used. It is also important that colors be linearized before processing, which is the case for *all* image matting algorithms. Papers that generate ground truth alpha mattes for evaluation purposes normally use these techniques to obtain accurate matte estimates (Chuang *et al.* 2001, Wang and Cohen 2007b, Levin *et al.* 2008, Rhemann *et al.* 2008b). Exercise 9.9 has you do this as well.

Difference matting

A related approach when the background is irregular but known is called *difference matting* (Wright 2006, Brinkmann 2008). It is most commonly used when the actor or object is filmed against a static background, e.g., for office videoconferencing or person tracking applications (Toyama *et al.* 1999), or to produce silhouettes for volumetric 3D reconstruction techniques §11.3.1 (Szeliski 1993, Seitz and Dyer 1997, Seitz *et al.* 2006). It can also be used with a panning camera where the background is composited from frames where the foreground has been removed using a *garbage matte* §13.2.1 (Chuang *et al.* 2002).

In the case where the foreground and background motions can both be specified with parametric transforms, high-quality mattes can be extracted using a generalization of triangulation matting (Wexler *et al.* 2002). When frames need to be processed independently, however, the results are often of poor quality (Figure 9.37). In such cases, using a pair of stereo cameras as input can dramatically improve the quality of the results (Criminisi *et al.* 2006, Yin *et al.* 2007).

9.4.2 Natural image matting

The most general version of image matting is when nothing is known about the background except, perhaps, for a rough segmentation of the scene into foreground, background, and unknown regions, which is known as the *trimap* (Figure 9.34b). Some recent techniques, however, relax this requirement and allow the user to just draw a few strokes or scribble in the image (Wang and Cohen 2005a, Wang *et al.* 2007, Levin *et al.* 2008, Rhemann *et al.* 2008b, Rhemann *et al.* 2008a) (Figures 9.40 and 9.41). Fully automated single image matting results have also been reported (Levin *et al.* 2008). The survey paper by Wang and Cohen (2007a) has detailed descriptions and comparisons of all of these techniques, a selection of which are described more briefly below.

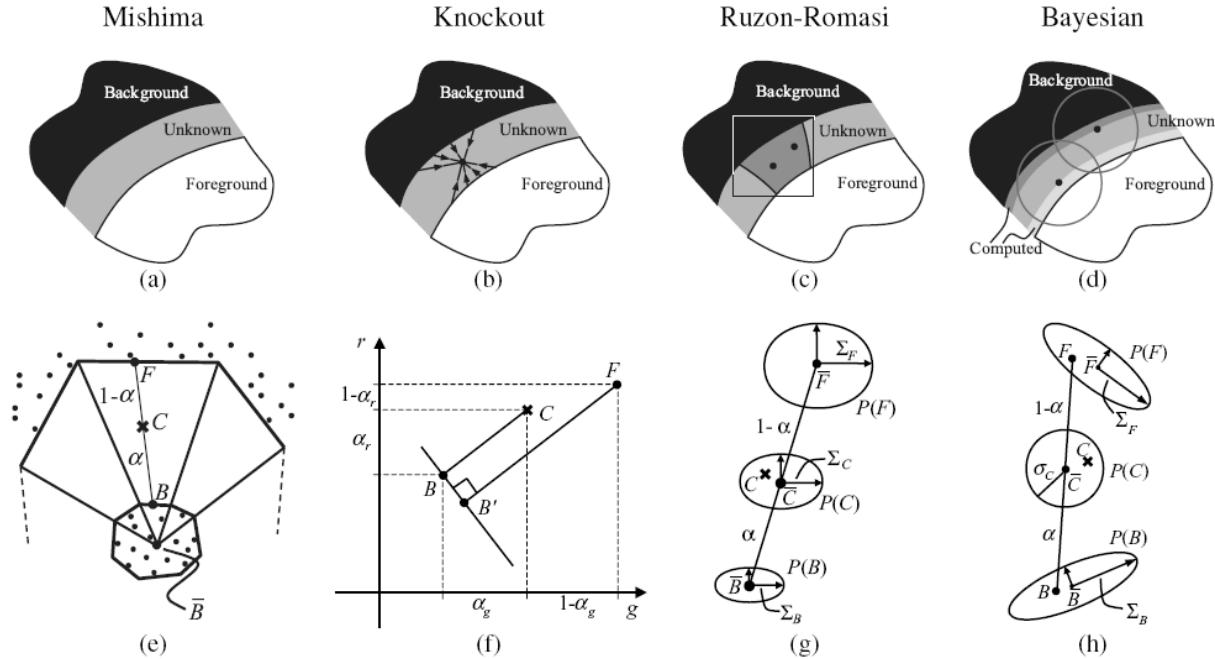


Figure 9.36: *Image matting algorithms* (Chuang et al. 2001). Mishima’s algorithm models global foreground and background color distribution as polyhedral surfaces centered around the mean background (blue) color. Knockout uses a local color estimate of foreground and background for each pixel and computes α along each color axis. Ruzon and Tomasi’s algorithm locally models foreground and background colors and variances. Chuang et al.’s Bayesian matting approach computes a MAP estimate of (fractional) foreground color and opacity given the local foreground and background distributions.

A relatively simple algorithm for performing natural image matting is Knockout, as described in (Chuang et al. 2001) and illustrated in Figure 9.36f. In this algorithm, the nearest known foreground and background pixels (in image space) are determined and then blended with neighboring known pixels to produce a per-pixel foreground F and background B color estimate. (The background color is then adjusted so that the measured color C lies on the line between F and B .) Opacity α is then estimated on a per-channel basis, and the three estimates are combined based on per-channel color differences. (This is an approximation to the least-squares solution for α .) Figure 9.37 shows that Knockout has problems when the background consists of more than one dominant local color.

More accurate matting results can be obtained if we treat the foreground and background colors as distributions sampled over some region (Figure 9.36g–h). Ruzon and Tomasi (2000) model local color distributions as mixtures of (uncorrelated) Gaussians and compute these models in strips.

They then find the pairing of mixture components F and B that best describes the observed color C , compute the α as the relative distance between these means, and adjust the estimates of F and B so they are collinear with C .

Chuang *et al.* (2001) and Hillman *et al.* (2001) use full 3×3 color covariance matrices to model mixtures of correlated Gaussians, and compute estimates independently for each pixel. Matte extraction proceeds in strips starting from known color values and proceeding inwards into the unknown region, so that recently computed F and B colors can be used in later stages.

To estimate the most likely value of an unknown pixel's opacity and (unmixed) foreground and background colors, Chuang *et al.* use a fully Bayesian formulation that maximizes

$$P(F, B, \alpha|C) = P(C|F, B, \alpha)P(F)P(B)P(\alpha)/P(C). \quad (9.27)$$

This is equivalent to minimizing the negative log likelihood

$$L(F, B, \alpha|C) = L(C|F, B, \alpha) + L(F) + L(B) + L(\alpha). \quad (9.28)$$

(The $L(C)$ term is dropped since it is constant.)

Let us examine each of these terms in turn. The first, $L(C|F, B, \alpha)$, is the likelihood that pixel color C was observed given values for the unknowns (F, B, α) . If we assume Gaussian noise in our observation with variance σ_C^2 , this negative log likelihood (data term) is

$$L(C) = \frac{1}{2} \|C - [\alpha F + (1 - \alpha)B]\|^2 / \sigma_C^2, \quad (9.29)$$

as illustrated in Figure 9.36h.

The second term, $L(F)$, correspond to the likelihood that a particular foreground color F comes from the mixture of Gaussians distribution. After partitioning the sample foreground colors into clusters, a weighted mean and covariance is computed, where the weights are proportional to a given foreground pixel's opacity and distance from the unknown pixel. The negative log likelihood for each cluster is thus given by

$$L(F) = (F - \bar{F})^T \Sigma_F^{-1} (F - \bar{F}). \quad (9.30)$$

A similar method is used to estimate unknown background color distributions. If the background is already known, i.e., for blue screen or difference matting applications, its measured color value and variance are used instead.

An alternative to modeling the foreground and background color distributions as mixtures of Gaussians is to keep around the original color samples and to compute most likely pairings that explain the observed color C (Wang and Cohen 2005a, Wang and Cohen 2007b). These techniques are described in more detail in (Wang and Cohen 2007a).

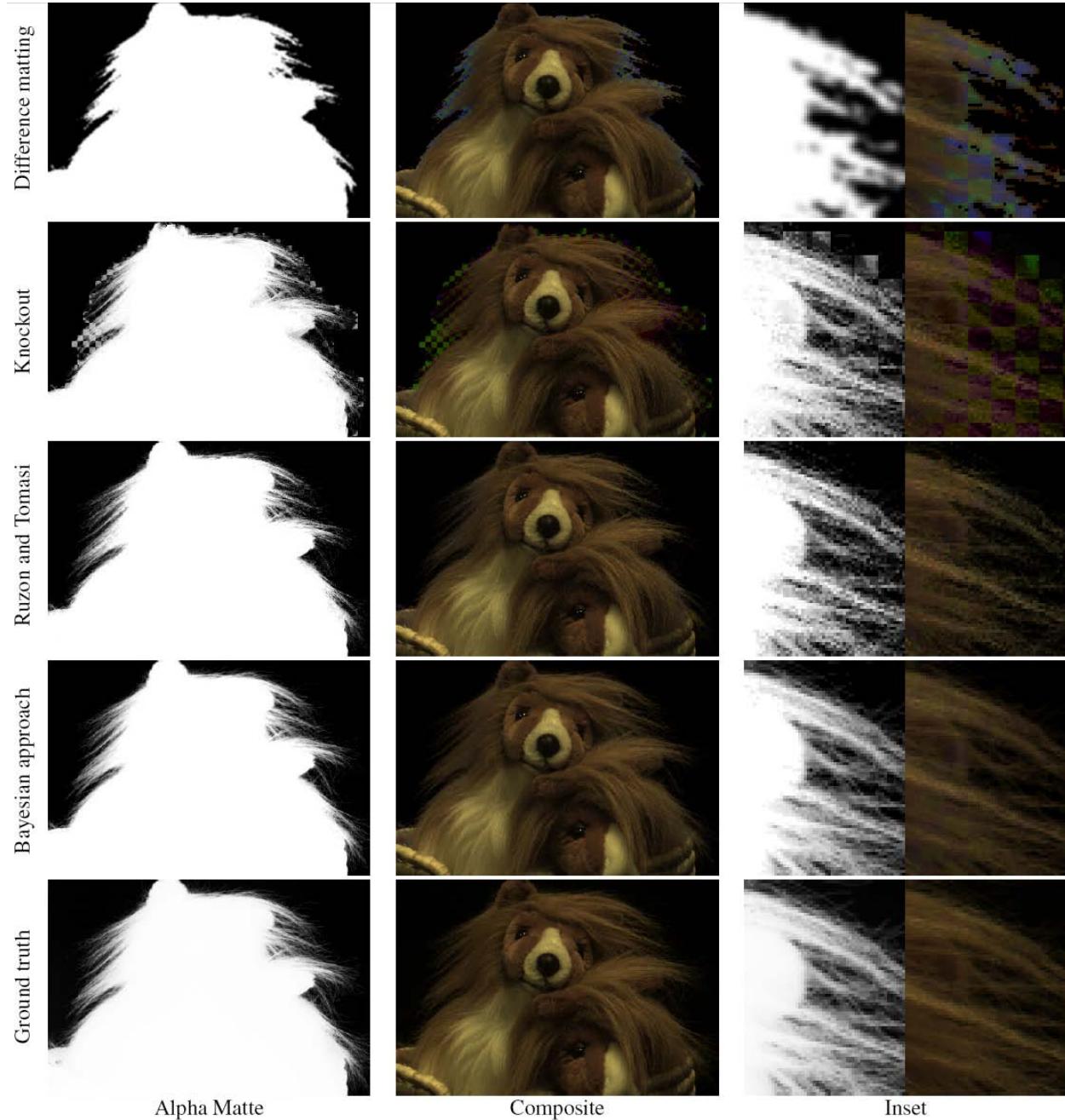


Figure 9.37: *Natural image matting results (Chuang et al. 2001)*. Difference matting and Knockout both perform poorly on this kind of background, while the more recent natural image matting techniques perform well. Chuang's result are slightly smoother and closer to the ground truth.

In their Bayesian matting paper, [Chuang *et al.* \(2001\)](#) assume a constant (non-informative) distribution for $L(\alpha)$. More recent papers assume this distribution to be more peaked around 0 and 1, or sometimes use Markov Random Fields (MRFs) to define a global (correlated) prior on $P(\alpha)$ ([Wang and Cohen 2007a](#)).

To compute the most likely estimates for (F, B, α) , the Bayesian matting algorithm estimate between computing (F, B) and α , since each of these problems is quadratic and hence can be solved as a small linear system. When several color clusters are estimated, the most likely pairing of foreground and background color clusters is used.

Bayesian image matting produces results that improve on the original natural image matting algorithm by [Ruzon and Tomasi \(2000\)](#), as can be seen in Figure 9.37. However, compared to more recent techniques ([Wang and Cohen 2007a](#)), its performance is not as good for complex background or inaccurate trimaps (Figure 9.39).

9.4.3 Optimization-based matting

An alternative to estimating each pixel's opacity and foreground color independently is to use global optimization to compute a matte that takes into account correlations between neighboring α values. Two examples of this are border matting in the Grab Cut interactive segmentation system ([Rother *et al.* 2004](#)) and Poisson Matting ([Sun *et al.* 2004](#)).

Border matting first dilates the region around the binary segmentation produced by Grab Cut §4.5.4 and then solves for a sub-pixel boundary location Δ and a blur width σ for every point along the boundary (Figure 9.33). Smoothness in these parameters along the boundary is enforced using regularization, and the optimization is performed using dynamic programming. While this technique can obtain good results for smooth boundaries such as a person's face, it has difficulty with fine details such as hair.

Poisson matting ([Sun *et al.* 2004](#)) assumes a known foreground and background color for each pixel in the trimap (as with Bayesian matting). However, instead of independently estimating each α value, it assumes that the gradient of the alpha matte and the gradient of the color image are related by

$$\nabla \alpha = \frac{F - B}{\|F - B\|^2} \cdot \nabla C, \quad (9.31)$$

which can be derived by taking gradients of both sides of (9.25) and assuming that the foreground and background vary slowly. The per-pixel gradient estimates are then integrated into a continuous $\alpha(x)$ field using the regularization (least squares) technique first described in §3.6.1 (3.99) and subsequently used in Poisson blending §8.3.3 (8.50) and gradient-based dynamic range compression mapping §9.2.1 (9.18). This technique works well when good foreground and background color estimates are available and these colors vary slowly.

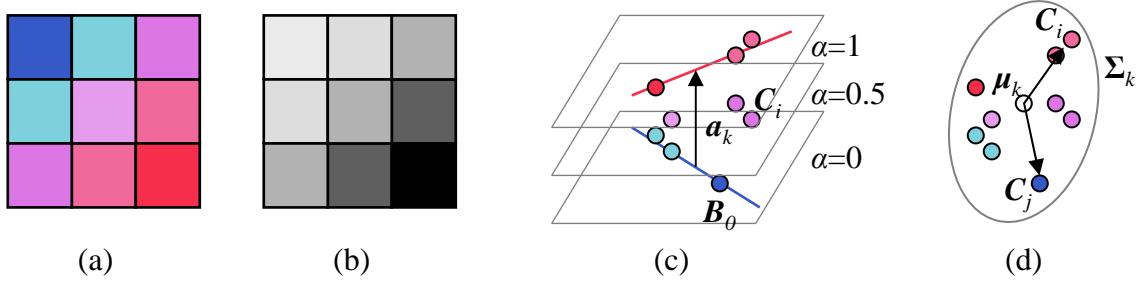


Figure 9.38: *Color line matting* (Levin et al. 2008): (a) local 3×3 patch of colors; (b) potential assignment of α values; (c) foreground and background color lines, the vector \mathbf{a}_k joining their closest points of intersection, and the family of parallel planes of constant α values, $\alpha_i = \mathbf{a}_k \cdot (\mathbf{C}_i - \mathbf{B}_0)$; (d) a scatter plot of sample colors and the deviations from the mean μ_k for two sample colors \mathbf{C}_i and \mathbf{C}_j .

Instead of computing per-pixel foreground and background colors, Levin et al. (2008) assume only that these color distribution can locally be well approximated as mixtures of two colors, which is known as the *color line model* (Figure 9.38a-c). Under this assumption, a closed-form estimate for α at each pixel i in a (say 3×3) window w_k is given by

$$\alpha_i = \mathbf{a}_k \cdot (\mathbf{C}_i - \mathbf{B}_0) = \mathbf{a}_k \cdot \mathbf{C} + b_k, \quad (9.32)$$

where \mathbf{C}_i is the pixel color treated as a 3-vector, \mathbf{B}_0 is any pixel along the background color line, and \mathbf{a}_k is the vector joining the two closest points on the foreground and background color lines, as shown in Figure 9.38c. (Note that the geometric derivation shown in this figure is an alternative to the algebraic derivation presented in (Levin et al. 2008).) Minimizing the deviations of the alpha values α_i from their respective color line models (9.32) over all overlapping windows w_k in the image gives rise to the cost

$$E_\alpha = \sum_k \left(\sum_{i \in w_k} (\alpha_i - \mathbf{a}_k \cdot \mathbf{C}_i - b_k)^2 + \epsilon \|\mathbf{a}_k\| \right), \quad (9.33)$$

where the ϵ term is used to regularize the value of \mathbf{a}_k in the case where the two color distributions overlap (i.e., in constant α regions).

Because this formula is quadratic in the unknowns $\{(\mathbf{a}_k, b_k)\}$, these can be eliminated inside each window w_k , leading to a final energy

$$E_\alpha = \mathbf{\alpha}^T \mathbf{L} \mathbf{\alpha}, \quad (9.34)$$

where the entries in the \mathbf{L} matrix are given by

$$L_{ij} = \sum_{k: i \in w_k \wedge j \in w_k} \left(\delta_{ij} - \frac{1}{M} \left(1 + (\mathbf{C}_i - \boldsymbol{\mu}_k)^T \hat{\Sigma}_k^{-1} (\mathbf{C}_i - \boldsymbol{\mu}_k) \right) \right), \quad (9.35)$$

where $M = |w_k|$ is the number of pixels in each (overlapping) window, μ_k is the mean color of the pixels in window w_k and $\hat{\Sigma}_k$ is the 3×3 covariance of the pixel colors plus $\epsilon_M \mathbf{I}$.

Figure 9.38d shows the intuition behind the entries in this affinity matrix, which is called the *matting Laplacian*. Note how when two pixels C_i and C_j in w_k point in opposite directions away from the mean μ_k , their weighted dot product is close to -1 , and so their affinity becomes close to 0. Pixels close to each other in color space (and hence with similar expected α values) will have affinities close to $-2/M$.

Minimizing the quadratic energy (9.34) constrained by the known values of $\alpha = \{0, 1\}$ at scribbles only requires the solution of a sparse set of linear equations, which is why the authors call their technique a *closed-form solution* to natural image matting. Once α has been computed, the foreground and background colors are estimated using a least-squares minimization of the compositing equation (9.25) regularized with a spatially-varying first order smoothness,

$$E_{B,F} = \sum_i \| \|C_i - [\alpha + F_i + (1 - \alpha_i)B_i]\|^2 + \lambda |\nabla \alpha_i| (\|\nabla F_i\|^2 + \|\nabla B_i\|^2), \quad (9.36)$$

where the $|\nabla \alpha_i|$ weight is applied separately for the x and y components of the F and B derivatives (Levin *et al.* 2008).

Laplacian (closed-form) matting is just one of many optimization-based techniques surveyed and compared in (Wang and Cohen 2007a). Some of these techniques use alternative formulations for the affinities / smoothness terms on the α matte, alternative estimation techniques such as belief propagation, or alternative representations (e.g., local histograms) for modeling local foreground and background color distributions (Wang and Cohen 2005a, Wang and Cohen 2007b, Wang and Cohen 2007c). Some of these techniques also provide real-time results as the user draws a contour line or sparse set of scribbles (Wang *et al.* 2007, Rhemann *et al.* 2008b) or even pre-segment the image into a small number of mattes that the user can select with simple clicks (Levin *et al.* 2008).

Figure 9.39 shows the results of running a number of the surveyed algorithms on a region of toy animal fur where a trimap has been specified, while Figure 9.40 show results for techniques that can produce mattes with only a few scribbles as input. Figure 9.41 shows a result for an even more recent algorithm (Rhemann *et al.* 2008b) that claims to outperform all of the techniques surveyed in (Wang and Cohen 2007a).

Pasting

Once a matte has been pulled from on image, it is usually composited directly over the new background, unless it is desired to hide the seams between the cutout and background regions, in which case Poisson Blending §8.3.3 (Pérez *et al.* 2003) can be used.

In the latter case, it is helpful if the matte boundary passes through regions that either have little texture or look similar in the old and new images. The Drag-and-Drop Pasting (Jia *et al.* 2006) and

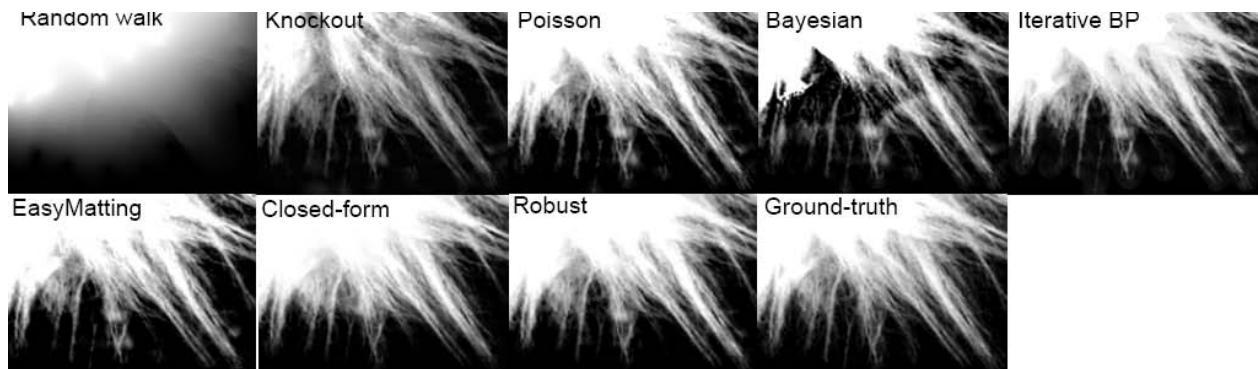


Figure 9.39: Comparative matting results for a medium accuracy trimap. See (Wang and Cohen 2007a) for a description of the individual techniques being compared.

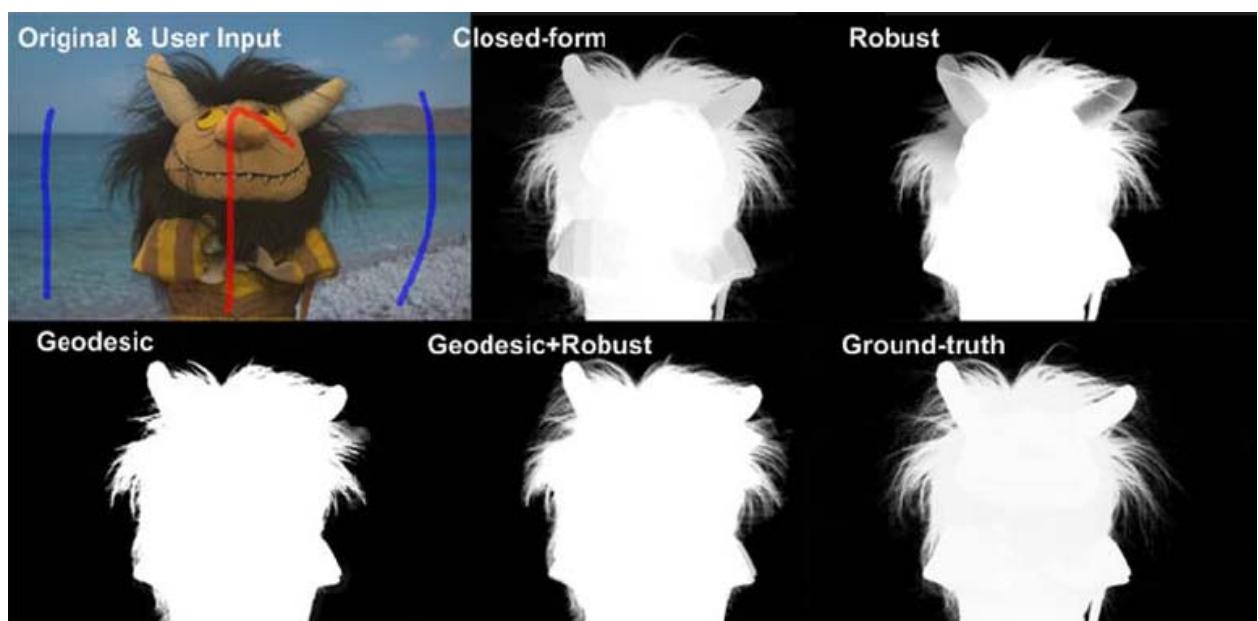


Figure 9.40: Comparative matting results with scribble-based inputs. See (Wang and Cohen 2007a) for a description of the individual techniques being compared.



Figure 9.41: Stroke-based segmentation result (Rhemann et al. 2008b):



Figure 9.42: *Smoke matting* ([Chuang et al. 2002](#)): (a) input video frame; (b) after removing the foreground object; (c) estimated alpha matte; (d) insertion of new objects into the background.

Simultaneous Matting and Compositing ([Wang and Cohen 2007c](#)) papers explain how to do this.

Smoke, shadow, and flash matting

In addition to matting out solid objects with fractional boundaries, it is also possible to matte out translucent media such as smoke ([Chuang et al. 2002](#)). Starting with a video sequence, each pixel is modeled as a linear combination of its (unknown) background color and a constant foreground (smoke) color that is common to all pixels. Voting in color space is used to estimate this foreground color, and the distance along each color line is used to estimate the per-pixel temporally varying alpha (Figure 9.42).

Extracting and re-inserting shadows is also possible using a related technique ([Chuang et al. 2003](#)). Here, instead of assuming a constant foreground color, each pixel is assumed to vary between its fully lit and fully shadowed colors, which can be estimated by taking (robust) minimum and maximum values over time as a shadow passes over the scene (Exercise 9.10). The resulting fractional *shadow matte* can be used to re-project the shadow into a novel scene. If the destination scene has non-planar geometry, it can be scanned by waving a straight stick shadow across the scene and then warping the shadow matte with the computed deformation field to have it drape correctly over the new scene (Figure 9.43).

The quality and reliability of matting algorithms can also be enhanced using more sophisticated acquisition systems. For example, taking a flash and non-flash image pair supports the reliable extraction of foreground mattes, which show up as regions of large illumination change between the two images ([Sun et al. 2006](#)). Taking simultaneous video streams focused at different distances ([McGuire et al. 2005](#)) or using multi-camera arrays ([Joshi et al. 2006](#)) are also good approaches to producing high-quality mattes. These techniques are described in more detail in ([Wang and Cohen 2007a](#)).

Lastly, photographing a refractive object in front of a number of patterned backgrounds allows the object to be placed in novel 3D environments. These environment matting techniques ([Zongker](#)

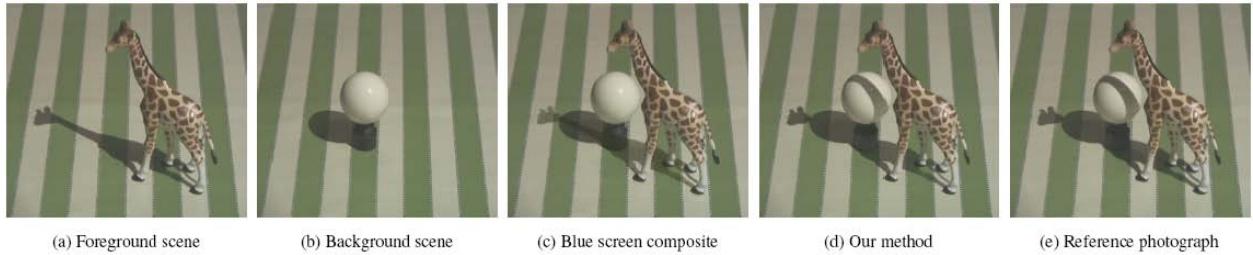


Figure 9.43: *Shadow matting* (Chuang et al. 2003). Instead of simply darkening the new scene with the shadow (c), shadow matting correctly dims the lit scene with the new shadow and drapes the shadow over 3D geometry (d).

et al. 1999, Chuang *et al.* 2000) are discussed in §12.4.

9.5 Texture analysis and synthesis

see Alyosha's VideoTexture.ppt notes for ideas and images

(De Bonet 1997) and earlier

(Efros and Leung 1999) and (Wei and Levoy 2000)

Texture quilting (Efros and Freeman 2001)

something by Song Chun Zhu ? <http://pascal.stat.ucla.edu:16080/> sczhu/publication.html

image-based priors for stereo (Fitzgibbon *et al.* 2003): belongs somewhere else

9.5.1 Application: Hole filling and inpainting

(Bertalmio *et al.* 2000, Bertalmio *et al.* 2003); (Criminisi *et al.* 2004) inpainting; (Efros and Leung 1999, Efros and Freeman 2001, Sun *et al.* 2004); (Komodakis and Tziritas 2007), LBP inpainting (Telea 2004): cited in OpenCV book

Instead of copying little pixels, can copy one whole image to fill a *big* hole: (Hays and Efros 2007).

9.6 Non-photorealistic rendering

Book: (Gooch and Gooch 2001), but now 7 years old

Web site: <http://www.cs.utah.edu/npr/papers.html>

NPAR conferences: <http://www.npar.org/>

Image analogies is one example, (Hertzmann *et al.* 2001). [Note: Newer variant on image analogies is (Cheng *et al.* 2008).]

Pen-and-ink ([Winkenbach and Salesin 1994](#)).

Painterly rendering ([Litwinowicz 1997](#)): see PDF for potential figures.

Stylization and Abstraction of Photographs ([DeCarlo and Santella 2002](#)) is another (see that work for more references, as well as the SIGGRAPH'06 paper on video-NPR). Our latest paper ([Farbman et al. 2008](#)) uses a different color abstraction algorithm to produce similar effects.

In general, converting a photo into something wow looking.

Most recent SIGGRAPH Asia 2008 papers: ([Kim et al. 2008](#), [Kolomenkin et al. 2008](#))

... any more topics? ...

[Note: Where does colorization ([Levin et al. 2004](#)) belong? It's a fun application and it should be covered. Perhaps in NPR?]

9.7 Exercises

Ex 9.1 (Radiometric calibration) Implement one of the multi-exposure radiometric calibration algorithms described in §9.2 ([Debevec and Malik 1997](#), [Mitsunaga and Nayar 1999](#), [Reinhard et al. 2005](#)). This calibration will be useful in a number of different applications, such as stitching images or stereo matching with different exposures and shape from shading.

1. Take series of bracketed images with your camera on a tripod. If your camera has an auto exposure bracket (AEB) mode, taking three images may be sufficient to calibrate most of your camera's dynamic range, especially if your scene has a lot of bright and dark regions. (Shooting outdoors or through a window on a sunny day is best.)
2. If your images are not taken on a tripod, first perform a global alignment (similarity transform).
3. Estimate the radiometric response function using one of the techniques cited above.
4. Estimate the high dynamic range radiance image by selecting or blending pixels from different exposures ([Debevec and Malik 1997](#), [Mitsunaga and Nayar 1999](#), [Eden et al. 2006](#)).
5. Repeat your calibration experiments under different conditions, e.g., indoors under incandescent light, to get a sense for the range of color balancing effects that your camera imposes.
6. If your camera supports RAW+JPEG mode, calibrate both sets of images simultaneously and to each other (the radiance at each pixel will correspond). See if you can come up with a model for what your camera does, e.g., whether it treats color balance as a diagonal or

full 3×3 matrix multiply, whether it uses non-linearities in addition to gamma, whether it sharpens the image while “developing” the JPEG image, etc.

7. Develop an interactive viewer to change the exposure of an image based on the average exposure of a region around the mouse. (One variant is to show the adjusted image inside a window around the mouse. Another is to adjust the complete image based on the mouse position.)
8. Implement a tone mapping operator (Exercise 9.5) and use this to map your radiance image to a displayable gamut.

Ex 9.2 (Noise level function) Determine your camera’s noise level function using either multiple shots or by analysing smooth regions.

1. Set up your camera on a tripod looking at a calibration target or a static scene with a good variation in input levels and colors. (Check your camera’s histogram to ensure that all values are being sampled.)
2. Take repeated images of the same scene (ideally with a remote shutter release) and average these to compute the variance at each pixel. Discarding pixels near high gradients (which are affected by camera motion), plot for each color channel the standard deviation at each pixel as a function of its output value.
3. Fit a lower envelope to these measurements and use this as your noise level function. How much variation do you see in the noise as a function of input level? How much of this is significant, i.e., away from flat regions in your camera response function where you don’t want to be sampling anyway?
4. [Optional:] Using the same images, develop a technique that segments the image into near-constant regions (Liu *et al.* 2008). (This is easier if you are photographing a calibration chart.) Compute the deviations for each region from a *single* image and use these to estimate the NLF. How does this compare to the multi-image technique, and how stable are your estimates from image to image?

Ex 9.3 (Vignetting) Estimate the amount of vignetting in some of your lenses using one of the following three techniques (or devise one of your choosing).

1. Take an image of a large uniform intensity region (well illuminated wall or blue sky—but be careful of brightness gradients) and fit a radial polynomial curve to estimate the vignetting.

2. Construct a center-weighted panorama and compare these pixel values to the input image values to estimate the vignetting function. Weight pixels in slowly varying regions more highly, as small misalignments will give large errors at high gradients. Optionally estimate the radiometric response function as well (Litvinov and Schechner 2005, Goldman and Chen 2005).
3. Analyze the radial gradients (especially in low gradient regions) and fit the robust means of these gradients to the derivative of the vignetting function, as described in (Zheng *et al.* 2008).

In all of these cases, be sure that you are using linearized intensity measurements, either by using RAW images, or images linearized through a radiometric response function, or at least images where the gamma curve has been removed.

- [Bonus question:] What happens if you forget to undo the gamma before fitting a (multiplicative) vignetting function?

For the parametric form of your vignetting function, you can either use a simple radial function, e.g.,

$$f(r) = 1 + \alpha_1 r + \alpha_2 r^2 + \dots \quad (9.37)$$

or one of the specialized equations developed in (Kang and Weiss 2000, Zheng *et al.* 2006).

Ex 9.4 (Optical blur (PSF) estimation) Compute the optical PSF using either a known target (Figure 9.7) or by detecting and fitting step edges §9.1.3 (Joshi *et al.* 2008).

1. Detect strong edges to sub-pixel precision.
2. Fit a local profile to each oriented edge and fill these pixels into an ideal target image, either at image resolution or at a higher resolution (Figure 9.9c–d).
3. Use least squares (9.1) at valid pixels to estimate the PSF kernel K , either globally or in locally overlapping sub-regions of the image.
4. Visualize the recovered PSFs, and then use these to remove chromatic aberration and/or de-blur the image (Exercise 9.8).

Ex 9.5 (Tone mapping) Implement one of the tone mapping algorithms discussed in §9.2.1, e.g., (Durand and Dorsey 2002, Fattal *et al.* 2002, Reinhard *et al.* 2002, Lischinski *et al.* 2006b) or any of the numerous additional algorithms discussed in (Reinhard *et al.* 2005) and <http://stellar.mit.edu/S/course/6/sp08/6.815/materials.html>.

Optionally compare your algorithm to local histogram equalization §3.1.4.

Ex 9.6 (Flash enhancement) Develop an algorithm to combine flash and non-flash photographs to best effect. You can use ideas from (Eisemann and Durand 2004, Petschnigg *et al.* 2004) or anything else you think might work well.

Ex 9.7 (Super-resolution) Implement a super-resolution algorithm: blind recovery (unknown transfer function);

calibrated transfer function (use circle/star test, raw readout camera), computed in Ex 9.4
non-linear prior (also related to blur/enhancement)

Ex 9.8 (Blur removal) Remove the blur in an image, using the PSFs estimated in Exercise 9.4. Implement one or more of the following techniques.

1. Generalized Wiener filtering: take the Fourier transform of the PSF and use this in (3.73)—oops! Not so fast! This equation doesn't model the blur kernel, just the signal power spectrum. Need to re-derive it (see notes from when Neel was interning), if it's not in his deconvolution paper.
2. Lucy-Richardson ...
3. non-linear prior, e.g., (Levin *et al.* 2007)

Ex 9.9 (Image matting) Develop an algorithm for pulling foreground matte from natural images, as described in §9.4.

1. Make sure that the images you are taking are linearized (Exercise 9.1 and §9.1) and that your camera exposure is fixed (full manual mode), at least when taking multiple shots of the same scene.
2. To acquire ground truth data, place your object in front of a computer monitor and display a variety of solid background colors as well as some natural imagery.
3. Remove your object and re-display the same images to acquire known background colors.
4. Use triangulation matting (Smith and Blinn 1996) to estimate the ground truth opacities α and pre-multiplied foreground colors αF for your objects.
5. Implement one of more of the natural image matting algorithms described in §9.4 and compare your results to the ground truth values you computed.
6. Optionally run your algorithms on other images taken with the same calibrated camera (or other images you find interesting).

Ex 9.10 (Smoke and shadow matting) Extract smoke and/or shadow mattes from one scene and insert them into another ([Chuang et al. 2002](#), [Chuang et al. 2003](#)).

1. Take a still or video sequence of images with and without some intermittent smoke and shadows. (Remember to linearize your images before proceeding with any computations.)
2. For each pixel, fit a line to the observed color values.
3. If performing smoke matting, robustly compute the intersection of these lines to obtain the smoke color estimate. Then, estimate the background color as the other extremum (unless you already took a smoke-free background image).
4. If performing shadow matting, compute robust shadow and lit (minimum and maximum) values for each pixel.
5. Extract the smoke or shadow mattes from each frame as the fraction between these two values (background and smoke or shadowed and lit).
6. Scan a new (destination) scene or modify the original background with an image editor.
7. Re-insert the smoke or shadow matte, along with any other foreground objects you may have extracted.
8. [Optional] Using a series of cast stick shadows, estimate the deformation field for the destination scene in order to correctly warp (drape) the shadows across the new geometry.
9. [Optional] [Chuang et al. \(2003\)](#) only demonstrated their technique for planar source geometries. Can you extend their technique to capture shadows acquired from an irregular source geometry?
10. [Optional] Can you change the direction of the shadow, i.e., simulate the effect of changing the light source direction?

Ex 9.11 (3D photography (revisited)) Enhance the system constructed in Ex 11.9, using a better texture extraction algorithms (undo shading and/or specularity, undo foreshortening). [*Note: Not sure what this means...delete if it doesn't make sense.*]

Ex 9.12 (Texture synthesis) Implement a texture synthesizer: Efros' is the simplest.

Ex 9.13 (Colorization) Implement the [Levin et al. \(2004\)](#) colorization algorithm. Find some old historic monochrome photographs, along with some modern color ones, and write an interactive tool that lets you “pick” colors from the modern photo and paint it over the old one. Tune the

algorithm parameters to give you good results. Are you please with the results? Can you think of ways to make them look more “antique”, i.e., with softer (less saturated and edgy) colors?

[Note: Where does colorization (Levin et al. 2004) belong? It’s a fun application and it should be covered.]

Chapter 10

Stereo correspondence

10.1	Epipolar geometry	462
10.1.1	Rectification	463
10.1.2	Plane sweep	465
10.2	Sparse correspondence	467
10.3	Dense correspondence	467
10.3.1	Similarity measures	468
10.4	Local methods	469
10.4.1	Sub-pixel estimation and uncertainty	470
10.4.2	<i>Application:</i> Stereo-based head tracking	471
10.5	Global optimization	472
10.5.1	<i>Application:</i> Z-keying (background replacement)	475
10.6	Multi-view stereo	476
10.6.1	Volumetric and 3D surface reconstruction	477
10.6.2	Shape from LightFields (<i>move?</i>)	478
10.7	Exercises	478

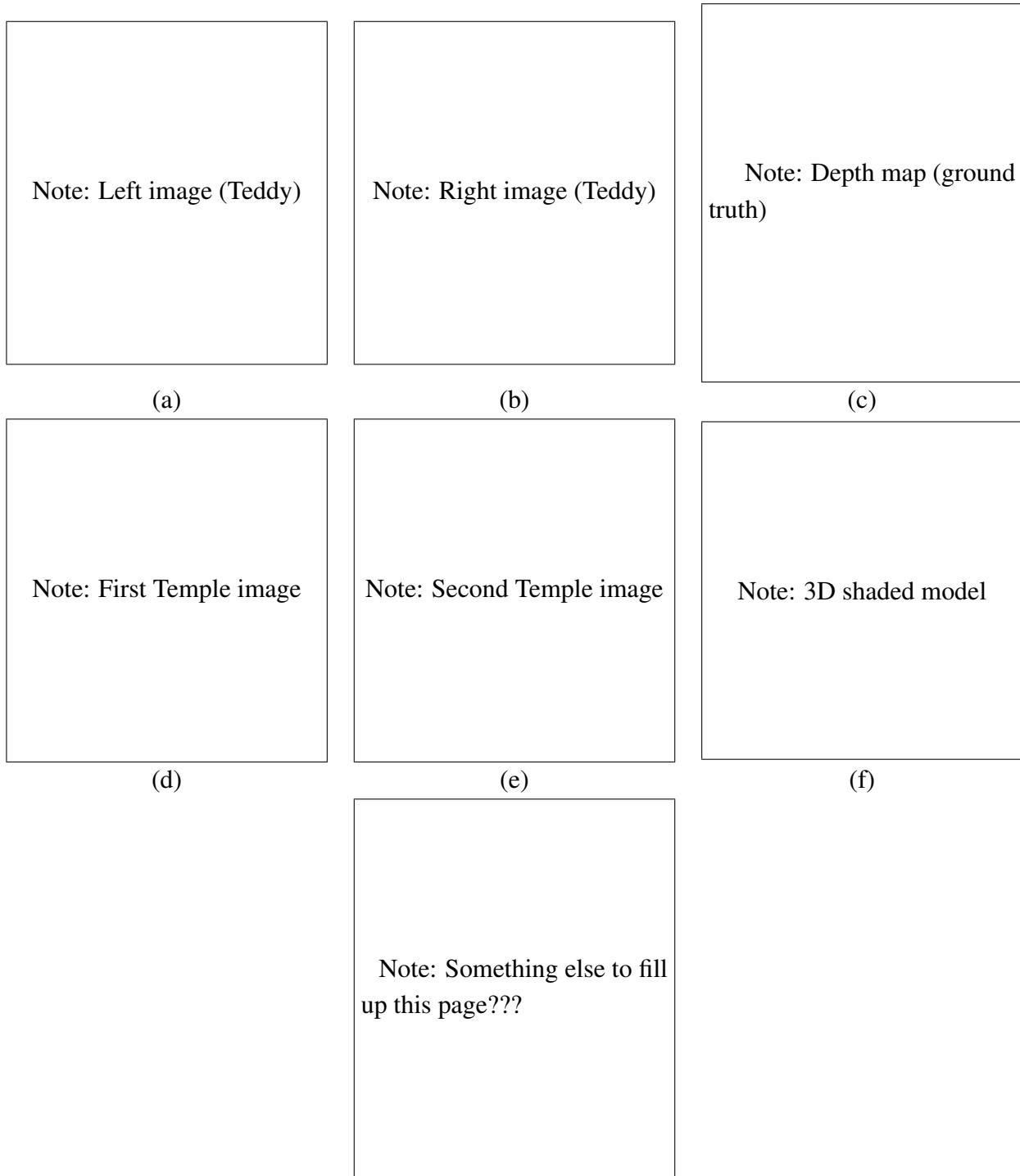
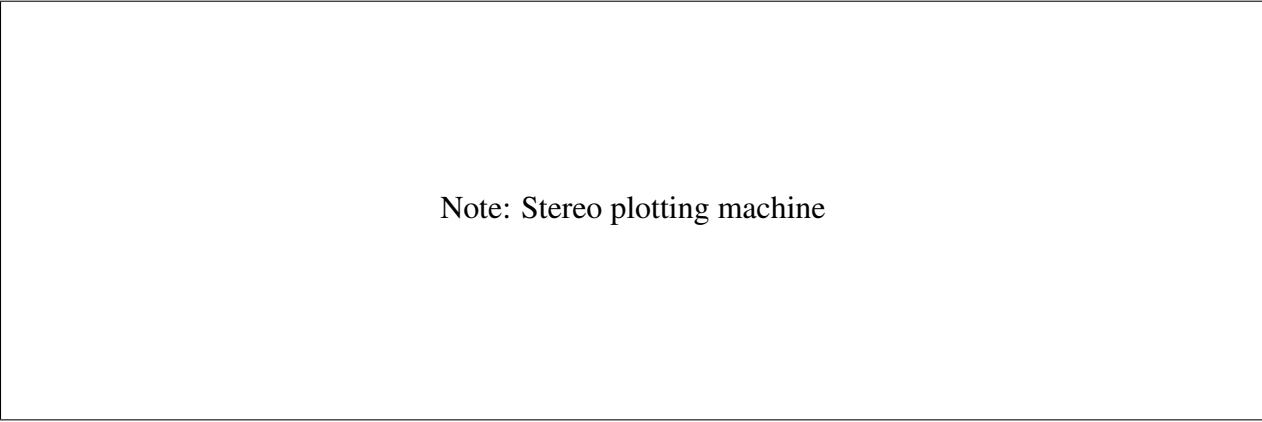


Figure 10.1: *Stereo reconstruction techniques can convert a pair of images (a–b) into a depth map (c), or a sequence of images (d–e) into a 3D model (f).*



Note: Stereo plotting machine

Figure 10.2: *Stereo plotting machine... TBD, get terminology right*

In previous chapters, we have developed techniques for recovering camera positions and building a sparse 3D model of a scene or object. In this chapter, we address the question of how to build a more complete 3D model of the scene. The next chapter §11.8 describes how to recover texture maps that can be painted onto the models to make them more realistic.

Why are people interested in stereo matching? From the earliest inquiries into visual perception, it was known that our visual system perceives depth based on the differences in appearance between the left and right eye.¹ As a simple experiment, hold your finger vertically in front of your eyes and close each eye alternately. You will notice that the finger jumps left and right relative to the background of the scene. The same phenomenon is visible in the image pair shown in Figure 10.1a–b, in which the foreground objects shift left and right relative to the background.

As we will shortly see, under simple imaging configurations (both eyes or cameras looking straight ahead), the amount of horizontal motion or *disparity* is inversely proportional to distance from the observer. (You can qualitatively confirm this for yourself by looking at a scene with a range of depths and alternately closing your eyes.) While the basic physics and geometry relating visual disparity to scene structure are well understood §10.1, automatically measuring this disparity by establishing dense and accurate inter-image *correspondences* is a challenging task.

The earliest stereo matching algorithms were developed in the field of *photogrammetry* for automatically constructing topographics elevation maps from overlapping aerial images. Prior to this (and even to this day), operators would use photogrammetric stereo plotters, which display shifted versions of such images to each eye, and allow the operator to float a dot cursor around constant elevation contours (Figure 10.2). The development of fully automated stereo matching algorithms was a major advance in this field, enabling much more rapid and less expensive exploitation of aerial imagery (Hannah 1974, Hsieh *et al.* 1992).

¹ The word *stereo* comes from the Greek for *solid*, which is how we perceive solid shape. [Note: Check this, and maybe look into (Koenderink 1990).]

In computer vision, the topic of stereo matching has been one of the most widely studied and fundamental problems (Marr and Poggio 1976, Barnard and Fischler 1982, Dhond and Aggarwal 1989, Scharstein and Szeliski 2002, Brown *et al.* 2003), and remains an active area of research. [Note: Read (Brown *et al.* 2003) and see what's there.] While photogrammetric matching concentrated mainly on aerial imagery, in computer vision, the range of applications includes modeling the human visual system (Marr 1982), robotic navigation and manipulation (Moravec 1983), 3D scene modeling for simulation and visualization (Kanade *et al.* 1997), and image-based rendering for visual effects. [Note: Add cite (last application) and pictures. Other potential applications: 3D head modeling for avatars and gaze correction, people tracking, other applications in this chapter... For view interpolation, can use (Matthies *et al.* 1989) figure or View Morphing (Seitz and Dyer 1996) used in my course note at Stereolf.ppt.]

In this chapter, I describe the fundamental principles behind stereo matching, following the general methodology proposed by Scharstein and Szeliski (2002). I begin in §10.1 with a review of the *geometry* of stereo image matching, i.e., how to compute for a given pixel in one image the range of possible locations the pixel might appear at in the other image (its *epipolar line*). I also describe how to pre-warp images so that corresponding epipolar lines are coincident (*rectification*), and then describe a general resampling algorithm called *plane sweep* that can be used to perform multi-image stereo matching with arbitrary camera configurations.

Next, I briefly survey techniques for *sparse* stereo matching of interest points and edge-like features §10.2. I then turn to the main topic of this chapter, namely the estimation of a *dense* set of pixel-wise correspondences in the form of a *disparity map* (Figure 10.1c). This involves first selecting a pixel matching criterion §10.3, and then using either local area-based aggregation §10.4 or global optimization §10.5 to help disambiguate potential matches. In the final part of this chapter §10.6, I discuss *multi-view stereo* methods that aim to reconstruct a complete 3D model instead of just a single disparity image (Figure 10.1d–f).

10.1 Epipolar geometry

[Note: Some of this may have already been introduced in §6.2.]

Given a pixel in one image (say the *left* image), how can we compute its correspondence with the correct pixel in the other image? In the chapter on motion estimation §7, we saw that a variety of search techniques can be used to match pixels based on their local appearance as well as the motions of neighboring pixels. In the case of stereo matching, however, we have some additional information available, namely the positions and calibration data for the cameras that took the pictures of the same (static) scene.

How can we exploit this information to reduce the number of potential correspondences, and

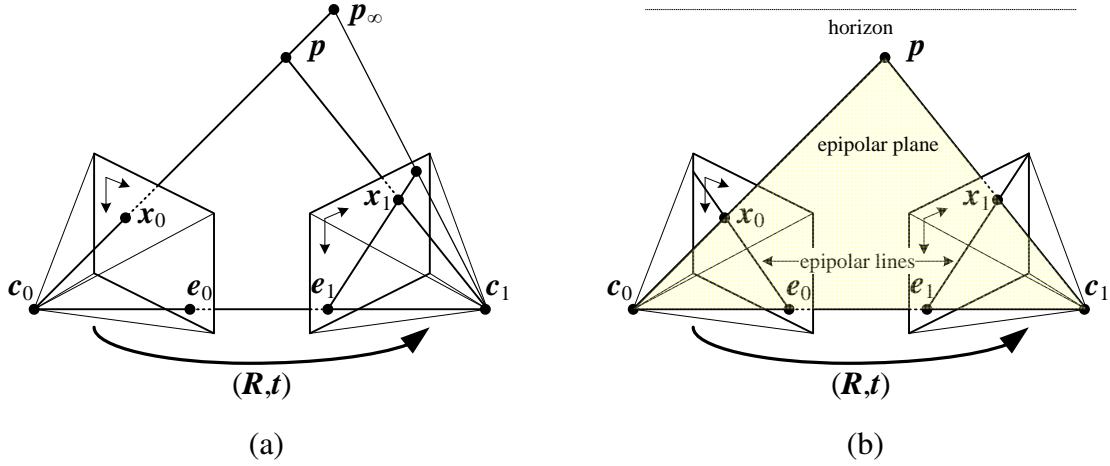


Figure 10.3: *Epipolar geometry:* (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane.

[Note: This figure may actually belong in the structure from motion section §6.]

hence both speed up the matching and increase its reliability? Figure 10.3a shows how a pixel in one image x_0 projects to an *epipolar line segment* in the other image. The segment is bounded at one end by the projection of the original viewing ray at infinity p_∞ and at the other end by the projection of the original camera center c_0 into the second camera, which is known as the *epipole* e_1 . If we project the epipolar line in the second image back into the first, we get another line (segment), this time bounded by the other corresponding epipole e_0 . Extending both line segments to infinity, we get a pair of corresponding *epipolar lines* (Figure 10.3b), which are the intersection of the two image planes with the *epipolar plane* that passes through both camera centers c_0 and c_1 as well as the point of interest p .

[Note: See also Sebastian's course notes, CS 223-B L5 Stereo [12].ppt.]

See (Faugeras and Luong 2001, Hartley and Zisserman 2004) for more detailed discussions of epipolar geometry.

10.1.1 Rectification

Once the epipolar geometry has been computed, we can use the epipolar line corresponding to a pixel in one image to constrain the search for corresponding pixels in the other image. One way to do this is to use a general correspondence algorithm such as optical flow §7.4, but to only consider locations along the epipolar line (and/or to project any flow vectors that fall off the line back onto it).

A more efficient algorithm can be obtained by first *rectifying* (i.e., warping) the input images so

that corresponding horizontal scanlines are epipolar lines.

[Note: Still struggling with smoothing all this together. Come back to it later.]

Why rectify images? Computing a separate epipolar line segment for each pixel and searching along that line is time consuming. Rectification warps images so that we have the usual “pure horizontal” disparity, which makes searching easier.

[Note: From taxonomy.tex:]

Recent references on stereo camera calibration and rectification include ([Zhang 1998a](#), [Loop and Zhang 1999](#), [Zhang 2000](#), [Hartley and Zisserman 2004](#), [Faugeras and Luong 2001](#)).

Basic idea: warp or rotate both images so that corresponding epipolar lines lie on the same image scanlines.

[Note: Still rough, smooth out:] If the two cameras have the same focal lengths, a simple way to do this is to rotate both cameras so that they are looking perpendicular to line joining the two camera centers c_0 and c_1 . Since there is a degree of freedom in the *tilt*, find the minimum rotation of the optical axes that achieves this. There still remains a possible twist around the optical axes. Make the *up vector* (camera y axis) perpendicular to the camera center line. This will ensure that corresponding epipolar lines are horizontal, and that the disparity for points at infinity is 0. [Note: Add a figure here, plus show top-down view as in ([Okutomi and Kanade 1993](#)).]

[Note: Make this an exercise, including deriving the appropriate rotation matrix. Optional: compare to the [Loop and Zhang \(1999\)](#) technique.]

This is the *standard rectified geometry* employed in a lot of stereo camera setups, and leads to a very simple inverse relationship between 3D depth and disparity ([Bolles et al. 1987](#), [Okutomi and Kanade 1993](#), [Scharstein and Szeliski 2002](#)).

[Note: The following text is copied from taxonomy.tex. Integrate it in with previous text.]

In computer vision, disparity is often treated as synonymous with inverse depth ([Bolles et al. 1987](#), [Okutomi and Kanade 1993](#)).

Since our goal is to compare a large number of methods within one common framework, we have chosen to focus on techniques that produce a univalued *disparity map* $d(x, y)$ as their output. Central to such methods is the concept of a *disparity space* (x, y, d) . The term *disparity* was first introduced in the human vision literature to describe the difference in location of corresponding features seen by the left and right eyes ([Marr 1982](#)). (Horizontal disparity is the most commonly studied phenomenon, but vertical disparity is possible if the eyes are verged.)

In this study [Note: fix wording], however, since all our images are taken on a linear path with the optical axis perpendicular to the camera displacement, the classical inverse-depth interpretation will suffice ([Okutomi and Kanade 1993](#)). The (x, y) coordinates of the disparity space are taken to be coincident with the pixel coordinates of a *reference image* chosen from our input data set. The correspondence between a pixel (x, y) in reference image r and a pixel (x', y') in matching image

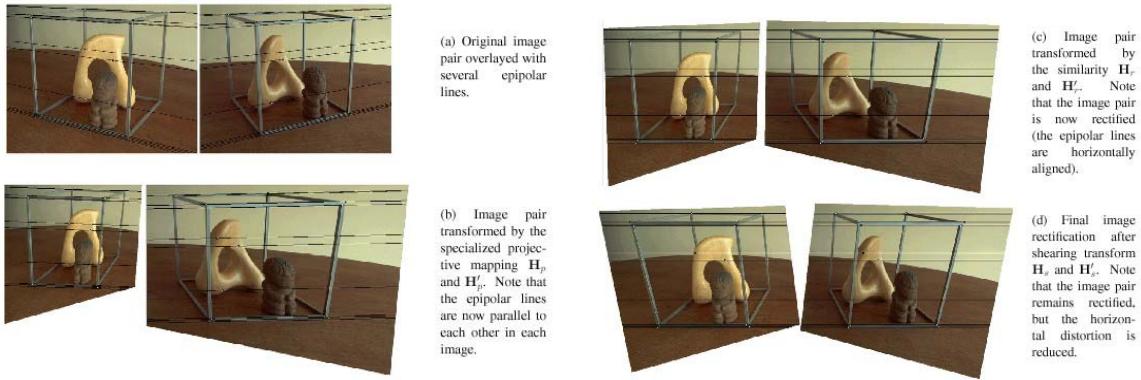


Figure 10.4: *An illustration of the multi-stage stereo rectification algorithm developed by Loop and Zhang.*

[Note: Rescan original images at better quality, get permissions.]

m is then given by

$$x' = x + s d(x, y), \quad y' = y, \quad (10.1)$$

where $s = \pm 1$ is a sign chosen so that disparities are always positive. Note that since our images are numbered from leftmost to rightmost, the pixels move from right to left.

Can be generalized to cameras all lying in a straight line (similar construction). For camera in a plane roughly perpendicular to their optical axes, can choose an image plane perpendicular to the camera plane and rescale the images. Then, epipolar lines are no longer all aligned, but homographies used in plane sweep 10.1.2 are all pure translations.

In the more general case, Loop and Zhang (1999) have developed a technique for first finding a homography that aligns corresponding scanlines, and then shears these images to keep the appearance of corresponding regions as similar as possible. Figure 10.4 shows the stages of this process.

10.1.2 Plane sweep

disparity space (x, y, d) is an old concept (Marr and Poggio 1976)

the name *disparity space image* was first coined by Intille and Bobick (1994);

plane sweeps (Collins 1996, Kanade *et al.* 1996, Szeliski and Golland 1999)

plane+parallax (Kumar *et al.* 1994b, Sawhney 1994b, Szeliski and Coughlan 1997): already discussed in Image Formation §2.1.4.

[Note: The following text is copied from taxonomy.tex. Move it to plane sweep, integrate it in:
]

More recently, several researchers have defined disparity as a three-dimensional projective

[Note: Get these figures and convert them to PDF:]

(a)	(b)	(c)	(d)	(e)
-----	-----	-----	-----	-----

[Note: Convert the following .eps file to .pdf: DS1a_s151.eps]

(f)

Figure 10.5: *Slices through a typical disparity space image (DSI): (a) original color image; (b) ground-truth disparities; (c–e) three (x, y) slices for $d = 10, 16, 21$; (e) an (x, d) slice for $y = 151$ (the dashed line in Figure (b)). Different dark (matching) regions are visible in Figures (c–e), e.g., the bookshelves, table and cans, and head statue, while three different disparity levels can be seen as horizontal lines in the (x, d) slice (Figure (f)). Note the dark bands in the various DSIs, which indicate regions that match at this disparity. (Smaller dark regions are often the result of textureless regions.)*

transformation (collineation or homography) of 3D space (X, Y, Z). The enumeration of all possible matches in such a generalized disparity space can be easily achieved with a *plane sweep* algorithm (Collins 1996, Szeliski and Golland 1999), which for every disparity d projects all images onto a common plane using a perspective projection (homography). (Note that this is different from the meaning of plane sweep in computational geometry.)

In general, we favor the more generalized interpretation of disparity, since it allows the adaptation of the search space to the geometry of the input cameras (Szeliski and Golland 1999, Saito and Kanade 1999); we plan to use it in future extensions of this work to multiple images. (Note that plane sweeps can also be generalized to other sweep surfaces such as cylinders (Shum and Szeliski 1999).)

Once the disparity space has been specified, we can introduce the concept of a *disparity space image* or DSI (Yang *et al.* 1993, Bobick and Intille 1999). In general, a DSI is any image or function defined over a continuous or discretized version of disparity space (x, y, d) . In practice, the DSI usually represents the confidence or log likelihood (i.e., *cost*) of a particular match implied by $d(x, y)$.

The goal of a stereo correspondence algorithm is then to produce a univalued function in disparity space $d(x, y)$ that best describes the shape of the surfaces in the scene. This can be viewed as finding a surface embedded in the disparity space image that has some optimality property, such as lowest cost and best (piecewise) smoothness (Yang *et al.* 1993). Figure 10.5 shows examples of slices through a typical DSI. More figures of this kind can be found in (Bobick and Intille 1999).

[Note: ...end of text from taxonomy.tex]

10.2 Sparse correspondence

Classic *feature-based* stereo: (Baker and Binford 1981, Grimson 1985, Ohta and Kanade 1985, Hsieh *et al.* 1992, Bolles *et al.* 1993) (see also edge-based and curve-based reconstruction, §11.2).

(Hsieh *et al.* 1992, Bolles *et al.* 1993) have comparisons of feature-based techniques.

More recent work by Zhang (Zhang and Shan 2000) and C. J. Taylor (Taylor 2003). Such sparse matching can also be useful for performing a free-space sweep (related to *volumetric reconstruction*, §11.3).

[Note: Not sure where to put in the hybrid techniques such as Joint View Triangulation (see <http://iris.usc.edu/Vision-Notes/bibliography/author/lhu.html>) or (Strecha *et al.* 2003) – reread these.
]

10.3 Dense correspondence

[Note: Give the taxonomy here (copy from taxonomy.tex):]

In order to support an informed comparison of stereo matching algorithms, we develop in this section a taxonomy and categorization scheme for such algorithms. We present a set of algorithmic “building blocks” from which a large set of existing algorithms can easily be constructed. Our taxonomy is based on the observation that stereo algorithms generally perform (subsets of) the following four steps (Scharstein and Szeliski 2002):

1. matching cost computation;
2. cost (support) aggregation;
3. disparity computation / optimization; and
4. disparity refinement.

The actual sequence of steps taken depends on the specific algorithm.

For example, *local* (window-based) algorithms, where the disparity computation at a given point depends only on intensity values within a finite window, usually make implicit smoothness assumptions by aggregating support. Some of these algorithms can cleanly be broken down into steps 1, 2, 3. For example, the traditional sum-of-squared-differences (SSD) algorithm can be described as:

1. the matching cost is the squared difference of intensity values at a given disparity;
2. aggregation is done by summing matching cost over square windows with constant disparity;

3. disparities are computed by selecting the minimal (winning) aggregated value at each pixel.

Some local algorithms, however, combine steps 1 and 2 and use a matching cost that is based on a support region, e.g. normalized cross-correlation (Hannah 1974, Bolles *et al.* 1993) and the rank transform (Zabih and Woodfill 1994). (This can also be viewed as a preprocessing step; see Section 10.3.1.)

On the other hand, *global* algorithms make explicit smoothness assumptions and then solve an optimization problem. Such algorithms typically do not perform an aggregation step, but rather seek a disparity assignment (step 3) that minimizes a global cost function that combines data (step 1) and smoothness terms. The main distinction between these algorithms is the minimization procedure used, e.g., simulated annealing (Marroquin *et al.* 1987, Barnard 1989), probabilistic (mean-field) diffusion (Scharstein and Szeliski 1998), or graph cuts (Boykov *et al.* 2001). [*Note: Add loopy belief propagation (Sun *et al.* 2003), EM? (Birchfield and Tomasi 1999a); look up all the newer techniques listed on the Middlebury site and reference the appropriate ones.*]

In between these two broad classes are certain iterative algorithms that do not explicitly state a global function that is to be minimized, but whose behavior mimics closely that of iterative optimization algorithms (Marr and Poggio 1976, Scharstein and Szeliski 1998, Zitnick and Kanade 2000). Hierarchical (coarse-to-fine) algorithms resemble such iterative algorithms, but typically operate on an image pyramid, where results from coarser levels are used to constrain a more local search at finer levels (Witkin *et al.* 1987, Quam 1984, Bergen *et al.* 1992).

10.3.1 Similarity measures

The first component of any dense stereo matching algorithm is a similarity measure that compares pixel values in order to determine how likely they are to be in correspondence. In this section, we briefly review the similarity measures we introduced in §7.1, and mention a few others that have been developed specifically for stereo matching.

[*Note: Text from taxonomy.tex:*]

The most common pixel-based matching costs include *squared intensity differences* (SD) (Hannah 1974) and *absolute intensity differences* (AD) (Kanade 1994). In the video processing community, these matching criteria are referred to as the *mean-squared error* (MSE) and *mean absolute difference* (MAD) measures; the term *displaced frame difference* is also often used (Tekalp 1995).

More recently, robust measures (7.2), including truncated quadratics and contaminated Gaussians have been proposed (Black and Anandan 1996, Black and Rangarajan 1996, Scharstein and Szeliski 1998). These measures are useful because they limit the influence of mismatches during aggregation. Vaish *et al.* (2006) compare a number of such robust measures, including a new one based on the entropy of the pixel values at a particular disparity hypothesis (which is particularly useful in multi-view stereo).

Other traditional matching costs include normalized cross-correlation (7.11) (Hannah 1974, Bolles *et al.* 1993), which behaves similar to sum-of-squared-differences (SSD), and binary matching costs (i.e., match / no match) (Marr and Poggio 1976), based on binary features such as edges (Baker and Binford 1981, Grimson 1985) or the sign of the Laplacian (Nishihara 1984). Binary matching costs are not commonly used in dense stereo methods, however.

Some costs are insensitive to differences in camera gain or bias, for example gradient-based measures (Seitz 1989, Scharstein 1994) and non-parametric measures such as rank and census transforms (Zabih and Woodfill 1994). The most general measure for compensating for inter-camera transforms is mutual information (Kim *et al.* 2003), which was first used popularized in the computer vision community for medical image registration (Viola and Wells III 1997). Of course, it is also possible to correct for different camera characteristics by performing a preprocessing step for bias-gain or histogram equalization (Gennert 1988, Cox *et al.* 1995). [Note: Dynamic histogram warping (Cox *et al.* 1995) matches image histograms using DP before stereo correspondence.] Invariance to gain and frequency-specific noise can also be achieved using phase and filter-bank responses (Marr and Poggio 1979, Kass 1988, Jenkin *et al.* 1991, Jones and Malik 1992).

In order to compensate for sampling issues, i.e., dramatically different pixel values in high-frequency areas, Birchfield and Tomasi (1998b) proposed a matching cost that is less sensitive to shifts in image sampling. Rather than just comparing pixel values shifted by integral amounts (which may miss a valid match), they compare each pixel in the reference image against a linearly interpolated function of the other image. A more detailed study of these ideas is explored in (Szeliski and Scharstein 2004).

[Note: end of text from taxonomy.tex]

10.4 Local methods

[Note: grab more text from taxonomy.tex ...]

[Note: Great survey and comparison of aggregation methods (including most recent ones from 2007) by Tombari *et al.* (2008).]

Local and window-based methods aggregate the matching cost by summing or averaging over a *support region* in the DSI $C(x, y, d)$. A support region can be either two-dimensional at a fixed disparity (favoring fronto-parallel surfaces), or three-dimensional in x - y - d space (supporting slanted surfaces). Two-dimensional evidence aggregation has been implemented using square windows or Gaussian convolution (traditional), multiple windows anchored at different points, i.e., shiftable windows (Arnold 1983, Bobick and Intille 1999), windows with adaptive sizes (Okutomi and Kanade 1992, Kanade and Okutomi 1994, Veksler 2001, Kang *et al.* 2001), and windows based

on connected components of constant disparity (Boykov *et al.* 1998). Three-dimensional support functions that have been proposed include limited disparity difference (Grimson 1985), limited disparity gradient (Pollard *et al.* 1985), and Prazdny's coherence principle (Prazdny 1985).

Aggregation with a fixed support region can be performed using 2D or 3D convolution,

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d), \quad (10.2)$$

or, in the case of rectangular windows, using efficient (moving average) box-filters. [Note: Are these covered in §3.2.1?] Shiftable windows can also be implemented efficiently using a separable sliding min-filter. [Note: Need to talk about separable sliding min filter: see implementation text in taxonomy.tex / sec:impl-aggr] A different method of aggregation is *iterative diffusion*, i.e., an aggregation (or averaging) operation that is implemented by repeatedly adding to each pixel's cost the weighted values of its neighboring pixels' costs (Szeliski and Hinton 1985, Shah 1993, Scharstein and Szeliski 1998).

In local methods, the emphasis is on the matching cost computation and on the cost aggregation steps. Computing the final disparities is trivial: simply choose at each pixel the disparity associated with the minimum cost value. Thus, these methods perform a local “winner-take-all” (WTA) optimization at each pixel. A limitation of this approach (and many other correspondence algorithms) is that uniqueness of matches is only enforced for one image (the *reference image*), while points in the other image might get matched to multiple points.

[Note: ... end taxonomy.tex text, start of loose notes]

Moving overlapped windows (Okutomi and Kanade 1993, Kanade and Okutomi 1994, Kanade *et al.* 1996, Kimura *et al.* 1999)

What window shape (and weight) to use? Horaud et al use double exponential.

Multi-window idea (Arnold 1983, Tao *et al.* 2001): can be implemented using a min-filter after aggregation

Aggregating over segmented regions (Yoon and Kweon 2005, Yoon and Kweon 2006)

Setting single disparity for color region (see also ICIP paper MP09.06: not sure where to find this?)

Teddy / Tao use Meer student's mean shift for segmentation (Tao *et al.* 2001). Also, Larry has used segmentation-based stereo (Zitnick *et al.* 2004). Or, does this belong below in global methods?

10.4.1 Sub-pixel estimation and uncertainty

[Note: Text from taxonomy.tex:]

Most stereo correspondence algorithms compute a set of disparity estimates in some discretized space, e.g., for integer disparities (exceptions include continuous optimization techniques such as

optical flow (Bergen *et al.* 1992) or splines (Szeliski and Coughlan 1997)). For applications such as robot navigation or people tracking, these may be perfectly adequate. However for image-based rendering, such quantized maps lead to very unappealing view synthesis results (the scene appears to be made up of many thin shearing layers). To remedy this situation, many algorithms apply a sub-pixel refinement stage after the initial discrete correspondence stage. (An alternative is to simply start with more discrete disparity levels (Szeliski and Scharstein 2004).)

Sub-pixel disparity estimates can be computed in a variety of ways, including iterative gradient descent and fitting a curve to the matching costs at discrete disparity levels (Ryan *et al.* 1980, Lucas and Kanade 1981, Tian and Huhns 1986, Matthies *et al.* 1989, Kanade and Okutomi 1994). This provides an easy way to increase the resolution of a stereo algorithm with little additional computation. However, to work well, the intensities being matched must vary smoothly, and the regions over which these estimates are computed must be on the same (correct) surface.

Recently, some questions have been raised about the advisability of fitting correlation curves to integer-sampled matching costs (Shimizu and Okutomi 2001). This situation may even be worse when sampling-insensitive dissimilarity measures are used (Birchfield and Tomasi 1998b). These issues are explored in more depth in (Scharstein and Szeliski 2002, Szeliski and Scharstein 2004).

Besides sub-pixel computations, there are of course other ways of post-processing the computed disparities. Occluded areas can be detected using cross-checking (comparing left-to-right and right-to-left disparity maps) (Cochran and Medioni 1992, Fua 1993). A median filter can be applied to “clean up” spurious mismatches, and holes due to occlusion can be filled by surface fitting or by distributing neighboring disparity estimates (Birchfield and Tomasi 1998a, Scharstein 1999).

[Note: ...end text from taxonomy.tex]

uncertainty analysis (Matthies *et al.* 1989) ((Szeliski 1991b) or (Carlbom *et al.* 1992) and Stereo1f.ppt have uncertainty images - reuse)

[Note: cross-checking (Fua 1993): good idea, be sure to cite!]

10.4.2 Application: Stereo-based head tracking

Used to be: Gaze correction (not gaze tracking). For desktop teleconferencing, interpolate mid-point view.

But this requires accurate 3D.

Use rough head position for “Fishtank VR” (3D parallax).

Can give gaze correction as additional application.

Describe Cox’s original idea (Ott *et al.* 1993), (“Teleconferencing Eye Contact Using a Virtual Camera,” M. Ott, J. P. Lewis and I. J. Cox, Interchi’93 Adjunct Proceedings, 109-110, Amsterdam, (1993), <http://www.ee.ucl.ac.uk/~icox/#Anchor--Ster-24671> - see also ACM Portal

)

newer CVPR'03 paper ([Criminisi *et al.* 2003](#)), with most recent work in ([Kolmogorov *et al.* 2006](#)).

Another application is people tracking (EasyLiving, in ([Toyama *et al.* 1999](#))? Also, Woodfill et al, many others...)

10.5 Global optimization

[Note: Text from taxonomy.tex; smooth over intro. In particular, energy based methods will already have been introduced in Image Processing / MRFs §3.6.2 and optical flow §7.3–7.4.]

In contrast, global methods perform almost all of their work during the disparity computation phase and often skip the aggregation step. Many global methods are formulated in an energy-minimization framework ([Terzopoulos 1986](#)). The objective is to find a disparity function d that minimizes a global energy,

$$E(d) = E_{\text{data}}(d) + \lambda E_{\text{smooth}}(d). \quad (10.3)$$

The data term, $E_{\text{data}}(d)$, measures how well the disparity function d agrees with the input image pair. Using the disparity space formulation,

$$E_{\text{data}}(d) = \sum_{(x,y)} C(x, y, d(x, y)), \quad (10.4)$$

where C is the (initial or aggregated) matching cost DSI.

The smoothness term $E_{\text{smooth}}(d)$ encodes the smoothness assumptions made by the algorithm. To make the optimization computationally tractable, the smoothness term is often restricted to only measuring the differences between neighboring pixels' disparities,

$$E_{\text{smooth}}(d) = \sum_{(x,y)} \rho(d(x, y) - d(x+1, y)) + \rho(d(x, y) - d(x, y+1)), \quad (10.5)$$

where ρ is some monotonically increasing function of disparity difference. (An alternative to smoothness functionals is to use a lower-dimensional representation such as splines ([Szeliski and Coughlan 1997](#))).

[Note: This whole survey thing may be too much for a textbook. Try to think of the main points, and cite the essential!]

In regularization-based vision ([Poggio *et al.* 1985](#)), ρ is a quadratic function, which makes d smooth everywhere and may lead to poor results at object boundaries. Energy functions that do not have this problem are called *discontinuity-preserving* and are based on robust ρ functions

(Terzopoulos 1986, Black and Rangarajan 1996, Scharstein and Szeliski 1998). Geman and Geman's seminal paper (Geman and Geman 1984) gave a Bayesian interpretation of these kinds of energy functions (Szeliski 1989) and proposed a discontinuity-preserving energy function based on Markov Random Fields (MRFs) and additional *line processes*. Black and Rangarajan (1996) show how line processes can be often be subsumed by a robust regularization framework.

The terms in E_{smooth} can also be made to depend on the intensity differences, e.g.,

$$\rho_d(d(x, y) - d(x+1, y)) \cdot \rho_I(\|I(x, y) - I(x+1, y)\|), \quad (10.6)$$

where ρ_I is some monotonically *decreasing* function of intensity differences that lowers smoothness costs at high intensity gradients. This idea (Gamble and Poggio 1987, Fua 1993, Bobick and Intille 1999, Boykov *et al.* 2001) encourages disparity discontinuities to coincide with intensity/color edges and appears to account for some of the good performance of global optimization approaches.

Once the global energy has been defined, a variety of algorithms can be used to find a (local) minimum. Traditional approaches associated with regularization and Markov Random Fields include continuation (Blake and Zisserman 1987), simulated annealing (Geman and Geman 1984, Marroquin *et al.* 1987, Barnard 1989), highest confidence first (Chou and Brown 1990), and mean-field annealing (Geiger and Girosi 1991).

More recently, *max-flow* and *graph-cut* methods have been proposed to solve a special class of global optimization problems (Roy and Cox 1998, Ishikawa and Geiger 1998, Boykov *et al.* 2001, Veksler 1999, Kolmogorov and Zabih 2001). Such methods are more efficient than simulated annealing and have produced good results. [Note: Add loopy belief propagation (Sun *et al.* 2003), TRW, maybe EM, and other algorithm from the MRF survey (Szeliski *et al.* 2008c).]

Dynamic programming. A different class of global optimization algorithms are those based on *dynamic programming*. While the 2D-optimization of Equation (10.3) can be shown to be NP-hard for common classes of smoothness functions (Veksler 1999), dynamic programming can find the global minimum for independent scanlines in polynomial time. Dynamic programming was first used for stereo vision in sparse, edge-based methods (Baker and Binford 1981, Ohta and Kanade 1985). More recent approaches have focused on the dense (intensity-based) scanline optimization problem (Belhumeur 1996, Geiger *et al.* 1992, Cox *et al.* 1996, Bobick and Intille 1999, Birchfield and Tomasi 1998a). These approaches work by computing the minimum-cost path through the matrix of all pairwise matching costs between two corresponding scanlines. Partial occlusion is handled explicitly by assigning a group of pixels in one image to a single pixel in the other image. Figure 10.6 shows one such example.

Problems with dynamic programming stereo include the selection of the right cost for occluded pixels and the difficulty of enforcing inter-scanline consistency, although several methods propose

[Note: Convert from .ps to .pdf]

Figure 10.6: *Stereo matching using dynamic programming.* For each pair of corresponding scanlines, a minimizing path through the matrix of all pairwise matching costs is selected. Lowercase letters (**a–k**) symbolize the intensities along each scanline. Uppercase letters represent the selected path through the matrix. Matches are indicated by **M**, while partially occluded points (which have a fixed cost) are indicated by **L** and **R**, corresponding to points only visible in the left and right image, respectively. Usually, only a limited disparity range is considered, which is 0–4 in the figure (indicated by the non-shaded squares). Note that this diagram shows an “unskewed” x - d slice through the DSI.

ways of addressing the latter (Ohta and Kanade 1985, Belhumeur 1996, Cox *et al.* 1996, Bobick and Intille 1999, Birchfield and Tomasi 1998a). Another problem is that the dynamic programming approach requires enforcing the *monotonicity* or *ordering constraint* (Yuille and Poggio 1984). This constraint requires that the relative ordering of pixels on a scanline remain the same between the two views, which may not be the case in scenes containing narrow foreground objects.

[Note: Mention combination with color segmentation (Kolmogorov *et al.* 2006)]

Cooperative algorithms. [Note: Move up? Some of the simplest kinds of optimization are these...] Finally, cooperative algorithms, inspired by computational models of human stereo vision, were among the earliest methods proposed for disparity computation (Dev 1974, Marr and Poggio 1976, Marroquin 1983, Szeliski and Hinton 1985). Such algorithms iteratively perform local computations, but use nonlinear operations that result in an overall behavior similar to global optimization algorithms. In fact, for some of these algorithms, it is possible to explicitly state a global function that is being minimized (Scharstein and Szeliski 1998). Recently, a promising variant of Marr and Poggio’s original cooperative algorithm has been developed (Zitnick and Kanade 2000).

[Note: ...end of text from taxonomy.tex]

Simulated annealing (Szeliski 1986, Marroquin *et al.* 1985, Barnard 1989)

Newer Swendsen-Wang (Barbu and Zhu 2003, Barbu and Zhu 2005)

See also (Boykov and Kolmogorov 2003) for better edge weightings (non-nearest neighbor).

Better segmentation results, but may be too complex to bother.

Belief propagation (Sun *et al.* 2003, Tappen and Freeman 2003): see stereo/taxonomy/schar.bib

[Note: Rationalize the discussion of MRFs with the Image Processing §3.6.2 and Appendix §B.5.]

Criminisi’s newest work on dynamic programming

Latest color segmentation work from Larry Zitnick (and references therein).

Recent CVPR'07 papers:

Learning Conditional Random Fields for Stereo, Daniel Scharstein and Chris Pal, ([Scharstein and Pal 2007](#)): Learning the right matching costs for stereo (nice work), but only marginal improvements (negative result).

Mumford-Shah Meets Stereo: Integration of Weak Depth Hypotheses. Thomas Pock, Christopher Zach, and Horst Bischof. ([Pock et al. 2007](#)): Elegant optimization framework that combines several low-level stereo as well as segmentation cues, and runs on a GPU.

Evaluation of Cost Functions for Stereo Matching. Heiko Hirschmüller and Daniel Scharstein. ([Hirschmüller and Scharstein 2007](#)): How well do stereo algorithms do when illumination changes? Nice comparison, with Heiko's semi-global algorithm plus mutual information performing well.

Recent CVPR'08 papers:

Classification and evaluation of cost aggregation methods for stereo correspondence. Federico Tombari, Stefano Mattoccia, Luigi Di Stefano and Elisa Addimanda. ([Tombari et al. 2008](#)): A very nice survey and evaluation of aggregation methods (variable window, segmentation, etc.) for stereo matching, along with software. The segmentation approaches introduced by Yoon and Kweon do best but are slow (18+ min). Veksler's integral image variable window is 3rd best and much faster (25 sec).

Global Stereo Reconstruction under Second Order Smoothness Priors. Oliver Woodford, Ian Reid, Phillip H.S. Torr and Andrew Fitzgibbon. ([Woodford et al. 2008](#)) Best Paper award! Add an L1 second order (horiz./vert. only) smoothness term, uses fusion moves with tilted plane proposals and QBPO.

Evaluation of Constructable Match Cost Measures for Stereo Correspondence Using Cluster Ranking. Daniel Neilson and Yee-Hong Yang. ([Neilson and Yang 2008](#)): An alternative way of ranking the algorithms on the Middlebury Stereo site which makes statistical significance explicit.

Illumination and Camera Invariant Stereo Matching. Yong Seok Heo, Kyoung Mu Lee and Sang Uk Lee. ([Heo et al. 2008](#)): Stereo matching under widely varying illumination conditions, which is currently an area of interest to Daniel.

10.5.1 Application: Z-keying (background replacement)

Show some z-keying results, and also virtual view generation. ([Kanade et al. 1996](#))

Criminisi's newer results using DP and color segmentation ([Kolmogorov et al. 2006](#)).

Forward reference to Virtual Viewpoint Video §13.5

Where to treat other techniques that do simultaneous stereo and matting such as ([Hasinoff et al. 2006](#)) and ([Taguchi et al. 2008](#))?

10.6 Multi-view stereo

[Note: Reread the survey (Seitz et al. 2006), as well as most recent papers from CVPR, e.g., (Hornung et al. 2008).]

[Note: Where to deal with multiple depth map approaches, e.g., (Szeliski 1999, Kang and Szeliski 2004, Maitre et al. 2008, Zhang et al. 2008) and multiple-input single depth map approaches (Kolmogorov and Zabih 2002, Kang and Szeliski 2004)?]

[Note: Some residual text grabbed from taxonomy.tex:]

The matching cost values over all pixels and all disparities form the initial disparity space image $C_0(x, y, d)$. While our study is currently restricted to two-frame methods, the initial DSI can easily incorporate information from more than two images by simply summing up the cost values for each matching image m , since the DSI is associated with a fixed reference image r (Equation (10.1)). This is the idea behind multiple-baseline SSSD and SSAD methods (Okutomi and Kanade 1993, Kang et al. 1995, Nakamura et al. 1996). As mentioned in §10.1.2, this idea can be generalized to arbitrary camera configurations using a plane sweep algorithm (Collins 1996, Szeliski and Golland 1999). [Note: Gallup et al. (2008) show how to adapt the baseline to the depth...]

Not all dense two-frame stereo correspondence algorithms can be described in terms of our basic taxonomy and representations. Here we briefly mention some additional algorithms and representations that are not covered by our framework.

The algorithms described in this paper first enumerate all possible matches at all possible disparities, then select the best set of matches in some way. This is a useful approach when a large amount of ambiguity may exist in the computed disparities. An alternative approach is to use methods inspired by classic (infinitesimal) optical flow computation. Here, images are successively warped and motion estimates incrementally updated until a satisfactory registration is achieved. These techniques are most often implemented within a coarse-to-fine hierarchical refinement framework (Quam 1984, Bergen et al. 1992, Barron et al. 1994, Szeliski and Coughlan 1997).

A univalued representation of the disparity map is also not essential. Multi-valued representations, which can represent several depth values along each line of sight, have been extensively studied recently, especially for large multi-view data set. Many of these techniques use a *voxel-based* representation to encode the reconstructed colors and spatial occupancies or opacities (Szeliski and Golland 1999, Seitz and Dyer 1999, Kutulakos and Seitz 2000, De Bonet and Viola 1999, Culbertson et al. 1999, Broadhurst et al. 2001). Another way to represent a scene with more complexity is to use multiple layers, each of which can be represented by a plane plus residual parallax (Baker et al. 1998, Birchfield and Tomasi 1999b, Tao et al. 2001). Finally, deformable surfaces of various kinds have also been used to perform 3D shape reconstruction from multiple images (Terzopoulos

and Fleischer 1988, Terzopoulos and Metaxas 1991, Fua and Leclerc 1995, Faugeras and Keriven 1998).

[Note: ...end text from taxonomy.tex]

epipolar plane image analysis (Bolles *et al.* 1987, Baker and Bolles 1989, Baker 1989)

incremental stereo (Matthies *et al.* 1989)

n-view stereo (Cox 1994, Szeliski 1999)

occlusion patterns (Nakamura *et al.* 1996)

multiple depth maps (Szeliski 1999)

spatio-temporal windows (Kang *et al.* 2001)

surface-based stereo (Fua and Leclerc 1995), described in more detail in §11.3.2.

Latest color segmentation work from Larry Zitnick: inter-frame consistency, multiple depth maps (Zitnick *et al.* 2004).

Robust multi-view and stereo vs. focus: (Vaish *et al.* 2006).

Non-planar manifolds

[Note: where to put this, exactly??]

Cylindrical stereo and multi-perspective (Ishiguro *et al.* 1992, Kang and Szeliski 1997, Shum and Szeliski 1999) (note: was not aware of (Ishiguro *et al.* 1992) until 11/20/2000)

10.6.1 Volumetric and 3D surface reconstruction

voxel coloring (Seitz and Dyer 1997) and space carving (Kutulakos and Seitz 1999, Culbertson *et al.* 1999, Saito and Kanade 1999, Eisert *et al.* 2000a)

transparent voxels (Szeliski and Golland 1999, De Bonet and Viola 1999)

Kolmogorov's multi-view stereo (Kolmogorov and Zabih 2002)

Probabilistic visibility for multi-view stereo, Carlos Hernandez, George Vogiatzis, and Roberto Cipolla, (Hernandez *et al.* 2007): Use match visibility (like CJ Taylor's work) to improve a volumetric graph cut multi-view stereo algorithm. Related to Brian Curless' signed distance map range merging.

A Surface-Growing Approach to Multi-View Stereo Reconstruction, Martin Habbecke and Leif Kobbelt, (Habbecke and Kobbelt 2007): Related to Michael Goesele's growing approach?

Accurate, Dense, and Robust Multi-View Stereopsis. Yasutaka Furukawa and Jean Ponce. (Furukawa and Ponce 2007): Very nice results, again related to Michael Goesele's approach. Yasu will come visit us in August.

S. Sinha, P. Mordohai, M. Pollefeys, Multi-View Stereo via Graph Cuts on the Dual of an Adaptive Tetrahedral Mesh, ICCV'07.

[Note: Put in more applications after the stereo section...]

Check out the latest papers from <http://vision.middlebury.edu/mview/>

Here's another one, not sure if it's relevant (not tested on latest data sets): ([Kolev et al. 2007](#)).

From CVPR08: ([Furukawa and Ponce 2008b](#), [Furukawa and Ponce 2008a](#))

10.6.2 Shape from LightFields (move?)

[Note: Surface lightfields are also in §12.3.2 in IBR chapter.]

[Note: does this belong in Shape Recovery §11? A lot of overlap with §11.8.1 Estimating BRDFs.]

Surface Light Fields ([Wood et al. 2000](#))

Shape from Synthetic Apertures ([Vaish et al. 2006](#)): already covered in Multi-View Stereo §10.6.

Simultaneously estimate 3D shape and BRDFs: ([Zhang et al. 2003](#), [Soatto et al. 2003](#))

10.7 Exercises

Ex 10.1 (Rigid direct alignment) Modify the spline-based motion estimator to use epipolar geometry (only estimate disparity).

Optional: estimating homography or plane equation

Ex 10.2 (Plane sweep) Implement a plane sweep algorithm. Try the following variants: equalization pre-processing; Birchfield/Tomasi's look for best match; square SSD and SAD windows; movable corner windows (min-filter)

Visualize the resulting DSI (use floating point image viewer)

Ex 10.3 (Window-based stereo) Apply a winner take-all to the DSI.

Optional: compute winners both ways, pick only reliable matches (draw others in another color)

Optional: tag matches that are unsure (too low confidence)

Optional: fill in the matches that are unsure

Ex 10.4 (Optimization-based stereo) Global optimization method (DP or graph cut or belief prop.).

Challenge: try to beat the best results on the Web

Ex 10.5 (View interpolation (revisited)) Compute dense depth map

View interpolation: forward mapping algorithm given in §3.5.2 and §12

Ex 10.6 (Multi-view stereo) Implement one of the multi-view algorithms (SSD with winnowing, voxel carving, Kolmogorov).

Challenge: try to get nice results on Flower Garden and Carousel sequence (use in-between prediction error as quality measure)

Ex 10.7 (Volumetric stereo) Write a voxel carving algorithm; explore different metrics for pixel similarity (original, Culbertson's, Kutulakos shuffle transform, Eisert's)

Chapter 11

Shape and appearance modeling

11.1	Shape from X	482
11.1.1	Shape from shading and photometric stereo	482
11.1.2	Shape from texture	482
11.1.3	Shape from focus	482
11.2	3D curves and profiles	482
11.3	Volumetric representations	483
11.3.1	Shape from silhouettes	483
11.3.2	Implicit surfaces and level sets (<i>move to multi-view stereo?</i>)	483
11.4	Active rangefinding	484
11.4.1	Range data merging	484
11.5	Surface representations	485
11.5.1	Surface interpolation	485
11.5.2	Surface simplification	486
11.5.3	<i>Application:</i> 3D photography	486
11.6	Point-based representations	486
11.7	Model-based reconstruction	487
11.7.1	Architecture	487
11.7.2	Heads and faces	488
11.7.3	Whole-body modeling and motion (<i>move to Motion?</i>)	488
11.8	Estimating texture maps and albedos (<i>move elsewhere?</i>)	489
11.8.1	Estimating BRDFs	489
11.9	Exercises	489

Recovering full 3D models (as opposed to depth maps).

Note that volumetric stereo §10.6.1 also does this, but was put in previous section because it has so much commonality with other stereo algorithms (e.g., plane sweep).

11.1 Shape from X

11.1.1 Shape from shading and photometric stereo

[Note: Downweight these older techniques. Give a brief intro and some math, but do not be exhaustive.]

Read the (Wolff *et al.* 1992b) collection for good material and possible cites.

(Pentland 1984, Horn and Brooks 1986, Horn and Brooks 1989, Horn 1990, Szeliski 1991a, Mancini and Wolff 1992, Dupuis and Oliensis 1994, Fua and Leclerc 1995, Zhang *et al.* 1999)

photometric stereo (Woodham 1981) (see also (Wolff *et al.* 1992b)): quite accurate, but still need to do normal integration; can be useful complement to stereo in textureless areas.

[Note: Recent paper by Harker and O'Leary (2008) that claims to do a better job of normal integration: uses discrete energy formulation (obvious?) and higher-order (polynomial) derivatives.
]

specularities: read some of the papers in (Wolff *et al.* 1992b)

multiplexed illumination (Schechner *et al.* 2003): may be even more useful in BRDF estimation
§11.8.1

Really nice work combining regular motion stereo/tracking with acquiring an environment map from specular reflections and then using it to refine the tracking (Lagger *et al.* 2008).

11.1.2 Shape from texture

(Witkin 1981, Malik and Rosenholtz 1997)

Include more recent cloth tracking and texture replacement, as well as Yanxi Liu's texture tracking and replacement.

11.1.3 Shape from focus

(Nayar *et al.* 1995)

11.2 3D curves and profiles

Edge and profile-based stereo:

Occluding contours and markings (Cipolla and Blake 1992, Vaillant and Faugeras 1992, Zheng 1994, Boyer and Berger 1997, Szeliski and Weiss 1998)

book: (Cipolla and Giblin 2000)

(Sullivan and Ponce 1998) goes from polyhedral silhouette to triangular spline (should this be in volume section, or at least a forward pointer?)

11.3 Volumetric representations

Voxels and octrees (Samet 1989): already used in volumetric stereo §10.6.1

Level sets and implicit surfaces (below)

11.3.1 Shape from silhouettes

Polyhedral approaches: which ones are (Potmesil 1987, Srivasan *et al.* 1990, Laurentini 1994)?

Octree representation (Samet 1989)

Octree approaches: which ones are (Potmesil 1987, Srivasan *et al.* 1990, Szeliski 1993, Laurentini 1994)?

Matusik & Buehler's visual hull (interval analysis) (Matusik *et al.* 2000) Matusik's latest silhouette based results

11.3.2 Implicit surfaces and level sets (*move to multi-view stereo?*)

Superquadrics (Pentland 1986, Solina and Bajcsy 1990, Waithe and Ferrie 1991),

Implicit functions as surface representations: (Lavallée and Szeliski 1995, Szeliski and Lavallee 1996, Frisken *et al.* 2000)

Poisson surface reconstruction (Kazhdan *et al.* 2006), which cites some other adaptive octree-based techniques [Grinspun *et al.* 2002; Losasso *et al.* 2004]. This could also go into surface interpolation, and even be mentioned in point-based reprs.

Turk's newest work

Level sets

(Faugeras and Keriven 1998)

Also, show how they can be used for inpainting (also mention for image restoration)

Surface-based stereo

Fua & Leclerc (Fua and Leclerc 1995) (uses Shape-from-Shading, previous section)
 newer work (Isidoro and Sclaroff 2003)

11.4 Active rangefinding

Besl's old survey (Besl 1989) from stereo/activeL/em.bib

Hebert2000 survey & others: find them

structured light: coded stripes, swept stripes (Rioux and Bird 1993, Curless and Levoy 1995, Bouguet and Perona 1998), projected grids (Proesmans *et al.* 1998), calibration using slant edge modulation (Goesele *et al.* 2003)

handheld 3D photography (shadow or laser stripe) (Bouguet and Perona 1998)

random illumination for better stereo matching (Kang *et al.* 1995) (apparently no journal version)

Space-Time stereo (Zhang *et al.* 2003)

Active Lighting stereo (Scharstein and Szeliski 2003)

11.4.1 Range data merging

The problem of fitting 3D models to sparse range data has been extensively studied in computer vision. Popular models include generalized cylinders (Agin and Binford 1976), superquadrics (Pentland 1986), and triangular meshes (Boissonat 1984). Physically-based models, which have internal deformation energies and can be fitted through external forces to 2D images and 3D range data have also been widely studied (Terzopoulos and Fleischer 1988, Terzopoulos *et al.* 1988, Delingette *et al.* 1991, Terzopoulos and Metaxas 1991, McInerney and Terzopoulos 1993, Terzopoulos 1999).

Range data registration: iterated closest point (ICP) (Besl and McKay 1992, Zhang 1994) and (signed) octree distance field (Lavallée and Szeliski 1995) (non-rigid version in (Szeliski and Lavallee 1996))

Range data merging: (Soucy and Laurendeau 1992, Turk and Levoy 1994, Hilton *et al.* 1996, Curless and Levoy 1996, Johnson and Kang 1997, Pulli 1999, Curless 1999)

run-length coding [Curless] for compactness ??

Digital Michelangelo: (Levoy *et al.* 2000)

New work on point cloud fitting and range data merging:

Global Optimization for Shape Fitting. Victor Lempitsky and Yuri Boykov. (Lempitsky and Boykov 2007) and references therein: Fit a set of oriented 3D data points with a closed surface

using an efficient variant of graph cut.

Shape from rotation (Szeliski 1991b)

11.5 Surface representations

general reference on curves and surfaces (Farin 1992, Farin 1996)

(also, check out these books from the library: The Nurbs Book (Monographs in Visual Communications) by Les A. Piegl, W. Tiller; Nurbs : From Projective Geometry to Practical Use by Gerald E. Farin ; Geometric Modeling by Michael E. Mortenson ; Curves and Surfaces in Geometric Design : Theory and Algorithms (Computer Graphics and Geometric Modeling) by Jean H. Gallier)

see also implicit functions and level set representations, below

11.5.1 Surface interpolation

regularization: (Terzopoulos 1986, Boult and Kender 1986, Szeliski 1990b, Nielson 1993)

Be sure to cover kernel / radial basis function solutions (aka *scattered data interpolation*) (Boult and Kender 1986, Nielson 1993)

Give the formula here, so people don't have to read (Nielson 1993): To interpolate a field $\mathbf{f}(\mathbf{x})$ through (or near) a number of data values \mathbf{z}_i located at \mathbf{x}_i , the *radial basis function* approach uses

$$\mathbf{f}(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x}) \mathbf{z}_i}{\sum_i w_i(\mathbf{x})}, \quad (11.1)$$

where the weights

$$w_i(\mathbf{x}) = K(\|\mathbf{x} - \mathbf{x}_i\|) \quad (11.2)$$

are computed using *radial basis functions* $K(d)$. [Note: Equation (3.95) uses d_i instead of \mathbf{z}_i . Make this consistent (drop the boldface and replace \mathbf{z} with d)?]

If we want the function $\mathbf{f}(\mathbf{x})$ to exactly interpolate the data points, the kernel functions must obey $\lim_{d \rightarrow 0} K(d) \rightarrow \infty$. It turns out that for certain regularized problems (3.93–3.95), there exist radial basis functions (kernels) that give the same results (Boult and Kender 1986), although the efficiency of such approaches decreases as the number of data points increases.

(Faugeras and Keriven 1998): does this belong here, or in volumetric?

Surface fitting to sparse data (Sander and Zucker 1990, Fua and Sander 1992, Hoppe *et al.* 1992); oriented particles (Szeliski and Tonnesen 1992, Szeliski *et al.* 1993a, Szeliski *et al.* 1993b); deformable surfaces (Delingette *et al.* 1991, McInerney and Terzopoulos 1993)

A nice application is Li Zhang's (Zhang *et al.* 2002) surface fitting to sparse normals with tears.

11.5.2 Surface simplification

Mesh simplification (Hoppe *et al.* 1993);
subdivision surfaces
triangular splines (Sullivan and Ponce 1998) (move elsewhere in chapter?)

11.5.3 Application: 3D photography

Mention Pollefeys' course

Track points, build silhouette shape, possibly profile, optionally build triangle mesh (Sullivan and Ponce 1998), texture map, use it to sell objects on eBay (VRML).

11.6 Point-based representations

unoriented particles and physically-based modeling

oriented particles (Szeliski and Tonnesen 1992, Szeliski *et al.* 1993b, Szeliski *et al.* 1993a)

newest work from Germany (Markus Gross? see seminar late Oct-03)

[*Note: Exercises now moved to the end of models.tex...*]

11.7 Model-based reconstruction

[Note: Used to be its own chapter, but collapsed into previous so there are less pure CV chapters.
]

11.7.1 Architecture

architecture (block) models – Facade *et al.* and Single View Metrology (Becker and Bove 1995,Debevec *et al.* 1996, Shum *et al.* 1998, Criminisi *et al.* 2000) + Tour Into the Picture (Horry *et al.* 1997)

(Antone and Teller 2000) and other Teller work

newest Cipolla / Torr work (using maps, parameterized window models, ...) (Dick *et al.* 2001, Robertson and Cipolla 2002), also related to VideoTrace (van den Hengel *et al.* 2007).

How about these: (Schaffalitzky and Zisserman 2000, Werner and Zisserman 2002, Mueller *et al.* 2007)

Grant Schindler's reconstructions (Schindler *et al.* 2008) (earlier cite to 4-D cities work?)

(Werner and Zisserman 2002): plane sweeps (cross-correlation at high gradient locations) in 3 orthogonal directions

Where should (Vergauwen and Van Gool 2006) Web-based reconstruction service go?

Urbanscan: (Pollefeys *et al.* 2008) and European version (Cornelis *et al.* 2008).

[Note: Complete TOC from that special issue: <http://www.springerlink.com/content/v42494464764/?p=161980e4df0348a48d785e066457fdf9&pi=6>

Modeling and Representations of Large-Scale 3D Scenes Zhigang Zhu and Takeo Kanade PDF (120.4 KB) 119-120

3D Urban Scene Modeling Integrating Recognition and Reconstruction Nico Cornelis, Bastian Leibe, Kurt Cornelis and Luc Van Gool PDF (1.8 MB) 121-141

Detailed Real-Time Urban 3D Reconstruction from Video M. Pollefeys, D. Nistr, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewnius, R. Yang, G. Welch and H. Towles PDF (2.0 MB) 143-167

Scanning Depth of Route Panorama Based on Stationary Blur Jiang Yu Zheng and Min Shi PDF (3.4 MB) 169-186

Minimal Aspect Distortion (MAD) Mosaicing of Long Scenes Alex Rav-Acha, Giora Engel and Shmuel Peleg PDF (2.9 MB) 187-206

Flying Laser Range Sensor for Large-Scale Site-Modeling and Its Applications in Bayon Digital Archival Project A. Banno, T. Masuda, T. Oishi and K. Ikeuchi PDF (1.5 MB) 207-222

Architectural Modeling from Sparsely Scanned Range Data Jie Chen and Baoquan Chen PDF (980.0 KB) 223-236

Integrating Automated Range Registration with Multiview Geometry for the Photorealistic Modeling of Large-Scale Scenes Ioannis Stamos, Lingyun Liu, Chao Chen, George Wolberg, Gene Yu and Siavash Zokai PDF (4.5 MB) 237-260

Building Illumination Coherent 3D Models of Large-Scale Outdoor Scenes Alejandro Troccoli and Peter Allen PDF (6.9 MB)

]

11.7.2 Heads and faces

priors on shape (head models)

(Pighin *et al.* 1998, Guenter *et al.* 1998, DeCarlo *et al.* 1998, Blanz and Vetter August 1999, Shan *et al.* 2001)

Kang's model-based s.f.m.; Zhengyou's 3D head modeler from video.

newer work? see Vetter, ICCV'03 (???) (Blanz and Vetter August 1999)

Fredo's notes 10_WarpMorph.pdf p. 72 has some nice photos

(Rowland and Perrett 1995) for early manual deformations, averaging, and shape + appearance (see Alyosha's notes, faces.ppt)

Face beautification (Leyvand *et al.* 2008)

Face swapping (Gross *et al.* 2008, Bitouk *et al.* 2008)

Face tracking: (Toyama 1998, Lepetit *et al.* 2005) + earlier publications by Kentaro on his thesis work?

Read the Lepetit and Fua (2005) survey...

How about Forsyth *et al.* (2005), Igarashi *et al.* (2007)?

PCA analysis (Turk and Pentland 1991b)

Synthesis of novel views from a single face image. [extended report PDF] T. Vetter, International Journal of Computer Vision, 28:2 (1998) 103-116. <http://gravis.cs.unibas.ch/publication.html>

Give pointer to whole face recognition literature, F&G workshops (newer name at ICCV'03).

Simon Baker et al, CVPR submission, PDFs/SimonBaker04

11.7.3 Whole-body modeling and motion (move to Motion?)

hands: (Rehg and Kanade 1994) (journal version?), (Blake and Isard 1998)

(Bregler and Malik 1998) (Cham and Rehg 1999), (Gleicher 1999)—see p51-gleicher.pdf, (Sidenbladh and Black 2003), (Andriluka *et al.* 2008)

(Rogez *et al.* 2008): randomized trees for 3D pose detection/estimation

mention mo-cap and mo-cap textures papers?

11.8 Estimating texture maps and albedos (move elsewhere?)

texture map recovery: mosaics (Szeliski and Shum 1997), face textures (Pighin *et al.* 1998), resolution issues (Wang *et al.* 2001)

Seamless Mosaicing of Image-Based Texture Maps. Victor Lempitsky and Denis Ivanov. (Lempitsky and Ivanov 2007): Run graph cuts over a 3D image-based mesh model to figure out which source images to use as textures, and then fix up the intensity discontinuities. Nice 3D generalization of Photomontage.

undo shading, c.f. shape-from-shading §11.1, (Sato and Ikeuchi 1996, Sato *et al.* 1997, Yu and Malik 1998, Yu *et al.* 1999, Debevec *et al.* 2000)

Rushmeier et al. paper [Note: get it back from Brian, or check in TVCG]

11.8.1 Estimating BRDFs

(Debevec *et al.* 2000) and Surface Light Fields (Wood *et al.* 2000)

multiplexed illumination (Schechner *et al.* 2003)

inverse global illumination (radiosity) (Yu *et al.* 1999)

11.9 Exercises

Need at least one good rotational 3D data set, to use for s.f.m. tracking, then silhouettes, profiles, triangle-based reconstruction, stereo merging, final texturing, ...

Ex 11.1 (Shadow striping) handheld 3D photography (shadow or laser stripe) (Bouguet and Perona 1998)

Ex 11.2 (Shape from shading) solve for shape (using any technique); test image same as that for hand-held stripe (for comparison) get a bas-relief, or matte spray a flat-ish object

Ex 11.3 (Shape from focus) grab a series of focused images (for calibration, sweep target plane away from camera at fixed focus), get qualitative shape; try sharp-everywhere (find references)

Ex 11.4 (Shape from profiles) edge and profile-based stereo: extract edges, track them, turn into 3D curves; keep visibility info, so can compute half-spaces

Ex 11.5 (Shape from silhouettes) Build a silhouette-based volume reconstruction algorithm; preferred representation: octree. [Note: Possible data structure for octree (in Appendix): A.R.G.B: A: 0=empty, 255=full (color), else 32-bit index into next level for first child node.]

Ex 11.6 (Range data registration) implement ICP or octree-distance registration apply to narrow-baseline stereo pairs

Ex 11.7 (Range data merging) use pairwise stereo and/or volumetric and/or edge-based; implement Curless and Levoy (1996) and/or Hilton *et al.* (1996) signed distance function (can use octree, if already implemented)

Ex 11.8 (Surface extraction and simplification) surface extraction and modeling: extract surface from distance function; simplify using Hoppe's technique

Ex 11.9 (3D photography) Put all of the above techniques together.

Establish a texture parameterization (see Hoppe's work?)

Inverse texture-map the surface (see §11.8 and Ex 9.11)

Convert to a VRML model

Ex 11.10 (Architectural modeler) Build 3D models from data set (see if Cipolla will share):

Extract lines for orientations and s.f.m

Find surfaces, extract textures.

Convert to VRML or other model viewer.

Ex 11.11 (Face modeler) Include the UW generic head model data set on the CD, and enough software so that student can try to model their own head.

Ex 11.12 (Body tracker) Track blobs, use a kinematic chain (more work)

Chapter 12

Image-based rendering

12.1	View interpolation	492
12.1.1	View-dependent texture maps	492
12.1.2	<i>Application:</i> Photo Tourism	492
12.2	Layered depth images	492
12.2.1	Geometry images	492
12.2.2	Impostors, sprites, and layers	493
12.3	Lightfields and Lumigraphs	494
12.3.1	Unstructured Lumigraph	494
12.3.2	Surface lightfields	494
12.3.3	Concentric and manifold mosaics	495
12.3.4	<i>Application:</i> Synthetic re-focusing	495
12.4	Environment mattes	495
12.5	The modeling / rendering continuum	495
12.6	Exercises	496

General survey / monograph: (Kang 1999, Shum *et al.* 2007)

Describe early work (McMillan and Bishop 1995, Chen 1995, Shade *et al.* 1996, Lengyel and Snyder 1997, Torborg and Kajiya 1996, Shade *et al.* 1998) already covered in previous chapter.

Lumigraph (below) places this on a firm footing.

Graphics / IBR continuum (Kang *et al.* 2000) – see end of chapter

12.1 View interpolation

(Chen and Williams 1993, Laveau and Faugeras 1994, McMillan and Bishop 1995)

view morphing (Beier and Neely 1992, Seitz and Dyer 1996)

12.1.1 View-dependent texture maps

Move this somewhere else? No: it makes sense as a view-dependant version of sprites, preview of more general (unstructured) Lumigraph.

(Debevec *et al.* 1996, Pighin *et al.* 1998, Pulli *et al.* 1998, Debevec *et al.* 1998)

Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle
 View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data Proceedings of 8th Eurographics Workshop on Rendering, St. Etienne, France, June 1997 also Technical Report UW-CSE-97-04-01, University of Washington

from <http://www-graphics.stanford.edu/kapu/resume.html>

12.1.2 Application: Photo Tourism

Photo Tourism (Snavely *et al.* 2006, Snavely *et al.* 2008b) and Photosynth

Lazebnik Workshop Internet Vision (w/ Frahm?)

[Note: mention (Aliaga *et al.* 2003)? Probably cited in paper.]

Could also be later after impostors, but no cutouts here, so simple enough (just need projective texture mapping).

12.2 Layered depth images

LDI (Shade *et al.* 1998)

12.2.1 Geometry images

Hugues' work...

12.2.2 Impostors, sprites, and layers

opaque layers: rendering (Shade *et al.* 1996, Lengyel and Snyder 1997, Torborg and Kajiya 1996, Shade *et al.* 1998) and recovery (Baker *et al.* 1998, Torr *et al.* 1999, Birchfield and Tomasi 1999b);

plane+parallax (Kumar *et al.* 1994b, Sawhney 1994b, Szeliski and Coughlan 1997, Baker *et al.* 1998);

video compression: mention MPEG-4, show PSNR for flower-garden?

How about recovery? There's Wang & Adelson, Baker's work, getting the mattes right (video matting, Hasinoff, ...). Find where these are presented and give a pointer.

What about Sing Bing's painting depth layers and pop-up lightfields?

Transparent layers and reflections [Note: Give a pointer to §7.5.]

[Note: Old text from when book was about IBMR] The term *image-based modeling* did not originate in the computer vision community, even though most of the techniques described in this book have been developed there. Rather, it was invented by computer graphics researchers who needed accurate three-dimensional models derived from real imagery to supply the photorealism required in high-end graphics applications such as movie special effects (Debevec *et al.* 1996). Image-based modeling can be used as a front end for *image-based rendering* systems, where geometric models are combined with realistic imagery (either photographed or accurately rendered) to produce novel images or videos with a high degree of realism. [Note: *IBR* may have been introduced in (McMillan and Bishop 1995) or (Chen 1995)—check this.]

[Note: Read Kang *et al.* (2006) survey, McMillan and Gortler (1999), Debevec (1999) articles...]

12.3 Lightfields and Lumigraphs

Lightfields (Levoy and Hanrahan 1996) and Lumigraph (Gortler *et al.* 1996)

(earlier work on plenoptic function (Adelson and Bergen 1991))

Great survey by Levoy (2006): reread.

Database of lightfields at Stanford: <http://lightfield.stanford.edu/>. See Marc Levoy's e-mail from 5/29/2008 and decide how to incorporate.

(results on light field compression: (Magnor and Girod 2000) + work by Harry Shum and Jin Li)

Have a look at Fredo's slides, 19_WavefrontCoding_LightField.pdf

In general, where do we discuss the role of image-based modeling in video compression applications (see e.g., (Eisert *et al.* 2000b, Li *et al.* 2000))?

12.3.1 Unstructured Lumigraph

(Buehler *et al.* 2001)

12.3.2 Surface lightfields

(Wood *et al.* 2000)

closely related to BRDF estimation, but motivated by lightfields

12.3.3 Concentric and manifold mosaics

Easy to build: rotate on offset, use s.f.m. to estimate camera pose (rotation angle, relative pose), re-sample columns... (Shum and He 1999, Shum *et al.* 2002)

Run stereo algorithm for compression

(results on light field compression: (Magnor and Girod 2000) + work by Harry Shum and Jin Li)

multi-perspective panoramas (Rademacher and Bishop 1998) and manifold mosaics (Peleg and Herman 1997)

Also, very long panoramic scenes (Agarwala *et al.* 2006), plus earlier manual work by Román *et al.* (2004), and more fully automated pipeline with Lidar by Román and Lensch (2006).

12.3.4 Application: Synthetic re-focusing

See description and references for synthetic aperture photography in (Levoy 2006), e.g.,

R. Ng et al., Light Field Photography with a Hand-Held Plenoptic Camera, tech. report CTSR 2005-02, Stanford Univ., 2005. (is there a SIGGRAPH'2005 paper??)

M. Levoy et al., Light Field Microscopy, to be published in ACM Trans. Graphics , vol. 25, no. 3, 2006.

(Vaish *et al.* 2006)

Coded aperture techniques (Levin *et al.* 2007)

12.4 Environment mattes

(Zongker *et al.* 1999, Chuang *et al.* 2000)

12.5 The modeling / rendering continuum

Graphics / IBR continuum (Kang *et al.* 2000)

Higher dimensional lightfields

Levoy's taxonomy, relate to more recent research.

Also, lighting models.

Environment mattes are only real example to date, but fixed viewpoint.

Also, Debevec's samples of lighting over the face (still 4D).

12.6 Exercises

Ex 12.1 (Depth image rendering) Generate a “view interpolation” result by re-rendering a previously computed stereo depth map. Use either graphics mesh rendering, or the forward warper constructed in Ex 3.24, modified to convert disparities into displacements

Ex 12.2 (View morphing) Modify the previous warper to blend inverse sampled input images, i.e., to a dense view morph. (Note: instead of taking the color image associated with the depth map, use the depth map projected to the new view position to index original images, and blend them appropriately.)

Ex 12.3 (Facial view morphing) Take some facial images, find axis of symmetry, reflect them, run stereo, and animate (Seitz and Dyer 1996).

Ex 12.4 (Multi-view morphing) Extend your view morphing code to more than two images.

Compute a consensus depth map using multi-view stereo.

Blend the nearest two or 3 images, based on view proximity. (This is starting to sound a lot like view-dependent texture mapping...)

Ex 12.5 (Layered depth images) Extend your forward warper to have more than one depth per pixel (LDI).

For your data, use either a synthetic ray tracing, a layered reconstructed model, or a volumetric reconstruction.

Ex 12.6 (Not sure...) Find dominant motions in selected regions; solve for camera motion; solve for plane equations; paint reconstructed regions. [*Note: What was I thinking when I wrote this one?*]

Ex 12.7 (Transparent layer recovery) Do a transparent motion pull: use a robust parametric estimator (developed for mosaics), find min-composite, stabilize residuals, find max composite, solve constrained least squares.

Ex 12.8 (Depth painting) Sing Bing’s paper, also SIGGRAPH paper on image editing in 3D.

Ex 12.9 (View dependent texture mapping) Use a previously constructed 3D model, but instead of extracting a single texture, extract one map for each input view, and blend between them. (Debevec *et al.* 1996, Pighin *et al.* 1998)

[*Note: Try to get Fred’s old face data along with models, and distribute the data for 3D morphing exercises.*]

Ex 12.10 (Natural image matting—harder) Implement the natural image matting algorithm described in §3.1.3 and (Ruzon and Tomasi 2000).

Construct a handheld Lumigraph (object on turntable, restricted motion)

Construct a handheld concentric mosaic (outward looking)

Add more...

Chapter 13

Video-based rendering

13.1	Video enhancement	500
13.1.1	Video denoising	500
13.2	Visual effects	500
13.2.1	Video matting	500
13.2.2	Video morphing	501
13.3	Video-based facial animation	501
13.4	Video textures	501
13.4.1	<i>Application:</i> Animating pictures	501
13.5	3D Video	502
13.5.1	<i>Application:</i> Video-based walkthroughs	502
13.6	Exercises	502

13.1 Video enhancement

We have already seen some of these applications...

Stabilization

Stabilization ([§7.2.1](#))

De-interlacing

Frame interpolation

[Note: separate section somewhere on video compression? No: not a focus of this book]

Video compression: discuss mosaics (MPEG-4), 3D video coding, lightfield compression, ...

13.1.1 Video denoising

Estimate flow, conservatively. After registration (or in low motion / low texture areas), average the images to reduce noise.

Publications by Zhengyou?

Try it out on some grainy, low-light video or USB video.

13.2 Visual effects

13.2.1 Video matting

Blue-screen matting and compositing covered in theory in (Smith and Blinn 1996) and in practice in (Wright 2006, Brinkmann 2008).

See survey paper by Wang and Cohen (2007a) for a nice summary of natural video matting, including rotoscoping.

Video matting ([Chuang et al. 2002](#))

([Wang and Cohen 2005b](#))

Wire removal

Match move

Video editing through unwrap mosaics ([Rav-Acha et al. 2008](#))

13.2.2 Video morphing

(Beier and Neely 1992) (§7.5)

13.3 Video-based facial animation

Video rewrite (Bregler *et al.* 1997)

Photo-based facial animation (Pighin *et al.* 1998, Blanz and Vetter August 1999)

Video-based facial animation (adding effects to video) (Pighin *et al.* 1999): include facial relighting and enhancement

Unwrap mosaics (Rav-Acha *et al.* 2008) is another example of this, but also applies to regular (non-facial) video.

Video-based texture maps and compression (Guenter *et al.* 1998)

Newest Poggio SIGGRAPH work (look in SIGBIB) and older Beymer / Poggio...

Video-based stereo and animation (Zhang *et al.* 2004)

Talk about face beautification (Leyvand *et al.* 2008) somewhere, maybe in model-based reconstruction or morphing?

13.4 Video textures

(Schödl *et al.* 2000)

(Szummer and Picard 1996)

(Bar-Joseph *et al.* 2001)

(Soatto *et al.* 2001)

(Agarwala *et al.* 2005)

Newer Shum work?

(Zhong and Sclaroff 2003): When matting foreground objects against repetitive moving backgrounds (i.e., VideoTextures), use the Soatto's ARMA Dynamic Texture to model the background appearance, and use a robustified Kalman filter to predict each particular frame.

(Wang and Zhu 2003) Modeling Textured Motion: Particle, Wave and Sketch: new work in the general VideoTexture area, also do some cartoon replacement of video using some hand-drawn frames.

13.4.1 Application: Animating pictures

(Chuang *et al.* 2005)

Describe how to matte individual layers, then animate them, either with motion, or in 3D (Sing Bing's depth painting, MIT's SG2003 (2002?) photo editing paper)

13.5 3D Video

Virtualized reality (3D video) ([Kanade et al. 1997](#), [Moezzi et al. 1996](#))

Video View Interpolation (Virtual Viewpoint Video) ([Zitnick et al. 2004](#))

13.5.1 Application: Video-based walkthroughs

Environment modeling and capture.

[Note: mention ([Aliaga et al. 2003](#))? Earlier work uses a sparse collection of panoramas.]

Describe vision of an interactive photo-realistic 3D world, and our progress to date.

([Uyttendaele et al. 2004](#))

13.6 Exercises

[Note: Need to fill in...]

Chapter 14

Recognition

14.1 Face recognition	505
14.1.1 Eigenfaces	506
14.1.2 Active appearance models	506
14.2 Face and object detection	506
14.2.1 <i>Application:</i> Personal photo collections	507
14.3 Instance recognition	507
14.3.1 Geometric alignment	507
14.3.2 Large databases	508
14.3.3 <i>Application:</i> Location recognition	508
14.4 Category recognition	509
14.4.1 Bag of words	509
14.4.2 Part-based models	509
14.4.3 Recognition with segmentation	509
14.4.4 Learning	510
14.4.5 <i>Application:</i> Image search	510
14.5 Context and scene understanding	510
14.6 Recognition databases and test sets	511
14.7 Exercises	511

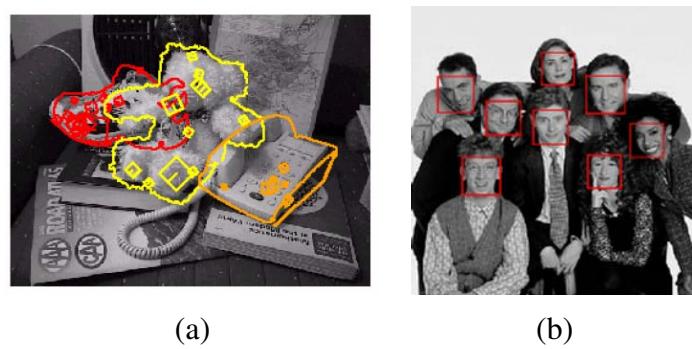


Figure 14.1: *Some examples of image stitching: (a) instance (known object) recognition (Lowe 1999); (b) real-time face detection (Viola and Jones 2004); (c) ...*

[Note: Constellation model: motorcycle or faces; Recognition with segmentation: pedestrian or cows; Context and tiny images: street scene, pop-up video, tiny images.]

From Steve's lecture notes: What is it? Object and scene recognition. Who is it? Identity recognition. Where is it? Object detection. What are they doing? Activities. All of these are classification problems: choose one class from a list of possible candidates.

A different taxonomy from [Csurka et al. 2006]: Recognition: Where is this particular object? Categorization: What kind of object(s) is(are) present? Content-based image retrieval: Find me something that looks similar. Detection: Locate all instances of a given class.

Another possible taxonomy: Direct matching and geometric alignment: Faces (eigenfaces and active appearance); 3D instance recognition: Lowe, Nister, Rothganger. Then do: Detection (faces and other stuff); Bag of words; Parts-based; ... in some order (detection can come early or late).

Organization of this chapter inspired by ICCV 2005 / CVPR 2007 short course on Recognizing and Learning Object categories ([Fei-Fei et al. 2007](#)) and also by the Taormina workshop book ([Ponce et al. 2007](#)).

Start with a general discussion of the classification problem, i.e., given a number of labeled training exemplars, label new data. See Steve's slides in `recog.ppt` on nearest neighbors and projections onto linear subspaces. Also, read and incorporate preliminary material from ([Bishop 2006](#)).

Give an outline of this chapter.

[Note: notes from CFAR NCAP meeting Dec 2008:

Chris Williams—databases: Pascal VOC, LabelMe, Caltech 101/256, Lotus Hill, Tiny Images, MSRC, NORB, BAe Soweby (old), Berkeley segmentation.

classification (is there a ? in the image): less interesting; localization: bounding box; segmentation: pixel labelling: class x vs. background annotated at the object level (different people); inpainting tasks: higher level knowledge for larger regions? Pascal Visual Object Classes (VOC) challenge; LabelMe: polygonal segm., non-standard labels, train vs. test.

Geoff: labelled Tiny images (120K); 10 categories, 5K train 1K test (like MNIST); 100 categories (distinct), 500 train 100 test. Learning MNIST with a bad supervisor: more bits is better. Nando: Yahoo has publically available labelled images with their BOSS program.]

14.1 Face recognition

Why faces first? Works “best” of all recognition tasks.

Take pictures of classmates in frontal views, similar illumination, can build near-perfect recognizer.

Harder if strong pose and illumination variation (PIE database). Mention face databases here or at end of section. (Find Eric Learned-Miller as well.)

Also harder if database is much larger, e.g., all people in facebook, or criminal databases, or attending superbowl. Basically, doesn’t work in this case, and even people aren’t that good at

recognizing beyond a few hundred known faces. [Note: Can you find a reference on human performance? Help, reviewers?]

Two main approaches: direct matching (same pose) and active appearance models (tolerant of expression and pose variation). [Note: Which techniques are doing the best right now? Ask Simon Baker and look in FERET and other databases.]

14.1.1 Eigenfaces

Eigenfaces (Turk and Pentland 1991b, Turk and Pentland 1991a): check which one to cite

Basic idea: project onto PCA space (explain relationship between PCA and SVD, here or in Appendix). Faster matching, potentially more accurate because projecting away irrelevant variation (although there's probably no real demonstration of this: Alex Berg claims that direct nearest neighbor is better than raw PCA).

How many components to use? Mostly a performance (speed) issue. It's possible to use cross-validation to set this, but many not show a strong optimum (don't believe it's been tried before).

Fisherfaces: (Belhumeur *et al.* 1997): find projections that are most discriminating. Called *linear discriminant analysis* in machine learning (check in (Bishop 1995, Bishop 2006)).

DFFS: separate projections for identity and expression (Moghaddam *et al.* 1996, Moghaddam and Pentland 1997).

Bayesian PCA: Tipping and Bishop + Roweis, as cited in (Torresani *et al.* 2008)

Eigenfaces with parts (Moghaddam *et al.* 1996, Moghaddam and Pentland 1997): check if journal is a superset; Bill Freeman's class notes cites a Pattern Recognition paper.

14.1.2 Active appearance models

Kanade's early (thesis) work: (Kanade 1977)

Active appearance (morphable) models:

(Rowland and Perrett 1995) (see Web page at <http://perception.st-and.ac.uk/Ageing/ageing.htm>; (Lanitis *et al.* 1995, Lanitis *et al.* 1997) (Beymer 1996) (Blanz and Vetter August 1999) (Matthews and Baker 2004, Matthews *et al.* 2007)

See nice (unpublished) survey in Cootes-AAM-TR04.pdf

14.2 Face and object detection

Can always scan a face recognizer over the image, but slow

Viola-Jones (Viola and Jones 2004); Boosting: see references in Torralba course, Bill Freeman's class. The book that Paul Viola used for machine learning?

Show a “perceptron” diagram with notes and arrows; mention that linear perceptron is just a matrix multiply; non-linear is classic “neural-network”. Boosting uses two-level decision (threshold) units.

From Torralba’s course (Torralba 2007): edge fragments: Opelt, Pinz, Zisserman ECCV 2006; shape contexts: Belongie... NIPS 2000; histogram of gradients: Dalal & Triggs 2006 (pedestrian detection)

Make pedestrian detection a subsubsection of face detection

Cascade of classifiers also attributed to Fleuret and Geman 2001 (co-discovery?)

Faster cascade learning by Jim Rehg and students, (Brubaker *et al.* 2008)

Shared features: Miller... CVPR 2000; Torralba, Murphy Freeman CVPR 2004 PAMI 2007

Other approaches:

neural networks: Rowley, Baluja, and Kanade 1995; also, convolutional neural nets (LeCun..)

SVMs: Heisele, Poggio, et al. NIPS 2001

Schneiderman 2004

Latest pedestrian (car, etc.) detection by Felzenszwalb *et al.* (2008), should have cites to earlier work such as (Dalal and Triggs 2005).

Maji *et al.* (2008) have super-fast detector that claims best result on INRIA pedestrian detection DB.

Another pedestrian detector/tracker, using randomized trees, is (Rogez *et al.* 2008). It produces full 3D pose information, closer to the (Sidenbladh and Black 2003, Andriluka *et al.* 2008) methods discussed in §11.7.3.

14.2.1 Application: Personal photo collections

Use face recognition (and clothes recognition) to index your personal photos by people.

(Mention other possibilities like location recognition §14.3.3 and activity/event recognition.)

14.3 Instance recognition

Recognizing instances of known objects and locations

14.3.1 Geometric alignment

See the CVPR 2007 course (Fei-Fei *et al.* 2007) for more historical cites and (Ponce *et al.* 2007) for Mundy’s historical review.

Mention that you can also use class- or instance-specific feature detectors that maximize *discriminability* from other classes (Ferencz *et al.* 2008).



Figure 14.2: *Recognizing objects in a cluttered scene* (Lowe 2004). Two of the training images in the database are shown on the left. These are matched to the cluttered scene in the middle using SIFT features, shown as small squares in the right image. The affine warping of each recognized database image onto the scene is shown as a larger parallelogram in the left image.

[Note: This same figure was used in the Feature Matching section §4.1.3.]

Lowe's edge-based approach

Newer feature-based approaches:

Lowe

(Kannala *et al.* 2008): a non-rigid extension of Lowe's technique, also does segmentation.

14.3.2 Large databases

Bag of words: (Sivic and Zisserman 2003) (and Csurka earlier?)

Vocabulary trees: (Nister and Stewenius 2006)

randomized forest of trees: (Philbin *et al.* 2007) say that these work better than vocabulary trees (has good citation for these algorithms)

Query expansion helps recall (Chum *et al.* 2007)

Latest work on soft quantization helps even more (Philbin *et al.* 2008).

14.3.3 Application: Location recognition

Match a personal photo (or cell-phone photo) against the large database of existing location photos (or within your own personal collection).

Mention Chum's latest TR on finding clusters (Chum and Matas 2008) and the Iconic Scene Graph work in this area (Li *et al.* 2008).

14.4 Category recognition

14.4.1 Bag of words

(Csurka *et al.* 2004, Lazebnik *et al.* 2006, Csurka *et al.* 2007, Zhang *et al.* 2007)

Survey paper in (Pinz 2005): read and discuss

Csurka was the first to introduce bag of keypoints

Lazebnik developed affine region descriptors (Lazebnik *et al.* 2005), and later extended Csurka's bag of keypoints to include spatial pyramids (Lazebnik *et al.* 2006). (Abstract says the technique offers insights into the performance of gist (Oliva and Torralba 2001, Torralba *et al.* 2003) and SIFT.)

Jurie's Extremely Randomized Clusters of Forests; Moosman, Triggs, Jurie, NIPS 06.

Good references to randomized forests in (Philbin *et al.* 2007).

Latest work from Michal Irani and students (Boiman *et al.* 2008) argues that not quantizing is even better (I guess at the expense of run-time): just compare features directly to all in-class features (only tested on a few Caltech 101 classes?)

Jamie Shotton's Semantic Texton Forests (Shotton *et al.* 2008b) do extremely well on the PASCAL VOC 2007 segmentation challenge.

You can also use class- or instance-specific feature detectors that maximize *discriminability* from other classes (Ferencz *et al.* 2008).

Combine codebook generation with classifier training (useful idea): (Yang *et al.* 2008).

General course notes: (Fei-Fei 2007)

Yann LeCun cites (Mutch and Lowe 2006) as 56% on CalTech 101; Yann's latest results, multiple stages with —tanh— non-linearities and final supervised refinement gets 65%

14.4.2 Part-based models

(Fergus 2007b)

Part-based: (Pentland *et al.* 1994, Burl and Perona 1996, Weber *et al.* 2000)

(Fergus *et al.* 2003, Felzenszwalb and Huttenlocher 2005, Fergus *et al.* 2005, Fei-Fei *et al.* 2006);

Elastic nets / pictorial structures (Felzenszwalb and Huttenlocher 2005)

Latest detector based on parts (Felzenszwalb *et al.* 2008)

What about grammars (Zhu and Mumford 2006)?

14.4.3 Recognition with segmentation

Malik believe segmentation is key to learning (find reference?)

Simultaneous recognition and segmentation (see CVPR course notes, (Fergus 2007a))

Simultaneous segmentation and recognition and CRFs (Mori *et al.* 2004, Shotton *et al.* 2006, Winn and Shotton 2006, Hoiem *et al.* 2007, Shotton *et al.* 2008a, Larlus and Jurie 2008).

Where to put this: (Malisiewicz and Efros 2008): over-segment the image, then find similar LabelMe images.

From Pushmeet Kohli's MSRC MRF Symposium slides: "Enforcing Label Consistency using Higher Order Potentials", (Kohli *et al.* 2008). References on Object Segmentation: [He *et al.* ECCV06, Yang *et al.* CVPR07, Rabinovich *et al.* ICCV07, Batra *et al.* CVPR08]

Rich Zemmel mentions a NIPS'2008 paper by Yuille on MSRC-23 pixel labelling (also cites Verbeek (sp?) and Triggs, Rich has his own work) - can't find NIPS preprints: ask Rich?

14.4.4 Learning

(Fergus *et al.* 2007, Fei-Fei *et al.* 2006)

(Felzenszwalb *et al.* 2008) best (?) results on PASCAL VOC07: from bbox, refine bbox to fit HOGs on grid, then learn 6 best sub-parts w/ finer-res HOGs and position distribution

(Varma and Ra 2007) uses multiple-kernel learning to pool other cues to get best ever VOC07 performance (see also upcoming (Bosch *et al.* 2008)).

How about (Fritz and Schiele 2008): find "topics" in 16x6 HOGs and then use these to do recognition (does well on VOC rigid objects)

Multiple segmentations combined with *hierarchical* appearance learning in (Sivic *et al.* 2008).

14.4.5 Application: Image search

Use context recognition software to enhance text-based image search.

Find some relevant images, see which ones appear the most similar (or match to a verified database), display the largest consistent cluster or several large clusters.

A variant (converse) of this is: given a query image, find similar LabelMe images (Malisiewicz and Efros 2008).

This is related to CBIR (Content Based Image Retrieval) and QBIC (Query by Image Content) (Flickner *et al.* 1995), two older fields that search primarily by simple image similarity metrics such as color and texture (Swain and Ballard 1991, Jacobs *et al.* 1995).

14.5 Context and scene understanding

See Antonio's slides (Torralba 2007) for some early work (Hanson and Riseman...)

Original context work (Torralba 2003, Torralba *et al.* 2003)

Murphy Torralba and Freeman, NIPS03; Torralba Murphy and Freeman, 2004; (*Hoiem et al. 2005*)

Context, hierarchies, and sharing: (*Sudderth et al. 2005*)

Tiny images: here or elsewhere - could almost be a direct matching technique. Read the paper first.

14.6 Recognition databases and test sets

From (*Fei-Fei et al. 2007*):

14.7 Exercises

Ex 14.1 (Recognition-based color balancing) Build a recognizer to recognize most important colors areas in common photographs (sky, grass, skin) and color balance the image accordingly.

Ex 14.2 (Think of something else) and add details here

Object detection / localization			
CMU/MIT frontal faces	vasc.ri.cmu.edu/idb/html/face/frontal_images cbcl.mit.edu/software-datasets/FaceData2.html	Patches	Frontal faces
Graz-02 Database	www.emt.tugraz.at/~pinz/data/GRAZ_02/	Segmentation masks	Bikes, cars, people
UIUC Image DB	l2r.cs.uiuc.edu/~cogcomp/Data/Cars/	Bounding boxes	Cars
TU Darmstadt DB	www.vision.ethz.ch/leibe/data/	Segmentation masks	Motorbikes, cars, cows
LabelMe dataset	people.csail.mit.edu/brussell/research/LabelMe/	Polygonal boundary	>500 Categories
Object recognition			
Caltech 101	www.vision.caltech.edu/Image_Datasets/Caltech101/	Segmentation masks	101 categories
Caltech 256			
COIL-100	www.cs.columbia.edu/CAVE/research/softlib/coil-100.html	Patches	100 instances
NORB	www.cs.nyu.edu/~ylclab/data/norb-v1.0/	Bounding box	50 toys
On-line annotation tools			
ETHZ Toys	www.robots.ox.ac.uk/~ferrari/datasets.html	?	toys and magazines?
ESP game	www.espgame.org	Image descriptions	Web images
PeekaBoom			
LabelMe	people.csail.mit.edu/brussell/research/LabelMe/	Polygonal boundary	High resolution images
Collections of challenges			
PASCAL	www.pascal-network.org/challenges/VOC/	Segmentation, boxes	various

Table 14.1: A list of image databases for recognition, taken from (Fei-Fei et al. 2007).

[Note: Yale face database at <http://www.cs.columbia.edu/~belhumeur/pub/images/yalefaces/readme> CMU PIE database. Also, “Faces in the Wild” labeled Internet faces at <http://vis-www.cs.umass.edu/lfw/>. Lotus Hill data set Benjamin Yao, Xiong Yang, and Song-Chun Zhu, “Introduction to a large scale general purpose ground truth dataset: methodology, annotation tool, and benchmarks.” EMMCVPR, 2007 <http://www.imageparsing.com/FreeDataOutline.html>. LabelMe has an associated journal paper, (Russell et al. 2008). Find a better reference to ETHZ toys (Ferrari’s Web page?). Check out A. Gallagher. Consumer image person recognition database, <http://amp.ece.cmu.edu/downloads.htm>, as cited in (Gallagher and Chen 2008).]

Chapter 15

Conclusions (or better title)

[Note: Most textbooks don't have such a section. Other possible titles: Summary, Epilogue, Prolegomena, ...]

Review what we have learned...

The modeling / rendering continuum, i.e., graphics / IBR continuum (Kang *et al.* 2000).

Pontificate on the future...

Modeling vs. data-driven (learning) approaches YES!!! This is clearly an emerging trend, and one that will become more pronounced. See David Salesin's NIPS invited talk.

In this book, we have approached the subject mostly from the “classical” approach of deterministic image processing (filtering, extraction, matching, reconstruction), although we have used estimation theory where applicable, and introduced Bayesian models. (See, e.g., how image enhancement works much better in the Bayesian (non-linear) domain...) These are sometimes overkill, but as we continue trying to aim for realism (video textures, video analogies) and more difficult inference problems (face hallucination, super-resolution), these will become more and more important.

Pull something on the merger of vision, graphics, and machine learning from U.T. slides...

Slow Glass

Appendix A

Linear algebra and numerical techniques

A.1	Matrix algebra	516
A.2	Linear regression	516
A.3	Non-linear regression	516
A.4	Sparse matrix techniques (direct)	516
A.5	Iterative techniques	517

A.1 Matrix algebra

some introductory material culled from (Golub and Van Loan 1996)

SVD, QR, Cholesky, eigenvalues

The eigenvalue decomposition of a symmetric semi-positive definite (SPD) matrix¹ C can be written as

$$\mathbf{A} = \mathbf{U} \operatorname{diag}(\lambda_0 \dots \lambda_{N-1}) \mathbf{U}^T, \quad (\text{A.1})$$

where $\lambda_0 \geq \lambda_1 \geq \dots \geq \lambda_{N-1} \geq 0$ are the *eigenvalues*, sorted in decreasing order, and the columns of \mathbf{U} are the corresponding *eigenvectors*.

condition numbers

A.2 Linear regression

Normal equations vs. QR (or SVD)

Singular Value Decomposition, least squares with QR

[Note: Also discuss Total Least Squares (TLS), (Van Huffel and Vandewalle 1991), for problems that have error in variables.]

A.3 Non-linear regression

Non-linear least-squares vs. iteratively reweighted least squares

Levenberg-Marquardt: using lambda

Also discuss other strategies, such as line search

Trust region methods (Conn *et al.* 2000)

A.4 Sparse matrix techniques (direct)

(Szeliski and Kang 1994, Triggs *et al.* 1999)

skyline method (Bathe 2007) [Note: what's the newer book I just bought?]

mention tri- and penta-diagonal and Toeplitz systems, which apply to snakes (4.35) and other 1-D problems.

re-read (Duff 1981) as well

¹ A symmetric semi-positive definite matrix (SPD) matrix is one for which $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$ for all \mathbf{x} .

A.5 Iterative techniques

different sparse representations: can express linearized equations, useful for fast multiplies in CG; how does it relate to Matlab sparse matrices?

related to *assembly* in finite elements (Bathe 2007)

gradient descent, conjugate gradient (Fletcher 1980, Gill *et al.* 1981, Shewchuk 1994) (re-read (Shewchuk 1994) carefully)

Trust region methods (Conn *et al.* 2000): covered in non-linear least squares section

multigrid (Briggs *et al.* 2000)

preconditioning, hierarchical bases (Szeliski 1990b, Pentland 1994, Szeliski 2006b); explain lifted wavelets again.

adaptive tree-based preconditioners (Koutis and Miller 2007, Koutis 2007, Koutis and Miller 2008, Grady 2008)

Appendix B

Estimation theory and Bayesian inference

B.1	Estimation theory	521
B.2	Maximum likelihood estimation and least squares	523
B.3	Robust statistics	524
B.4	Prior models and Bayesian inference	524
B.5	Markov Random Fields	525
B.6	Inference algorithms	525
B.6.1	Gradient descent and simulated annealing	525
B.6.2	Dynamic programming	525
B.6.3	Belief propagation	525
B.6.4	Graph cuts	526
B.7	Uncertainty estimation (error analysis)	527
B.8	Kalman filtering	527
B.9	Particle filtering	529

[Note: be sure to cite ([Bishop 2006](#)) somewhere.]

[Note: Copy some of this into intro as well?]

A commonly recurring problem in this book is the following. Given a number of measurements (images, feature positions, etc.), estimate the values of some unknown structure or parameter (camera positions, object shape, etc.) These kinds of problems are in general called *inverse* problems because they involve estimating unknown model parameters instead of simulating the forward formation equations.¹ Computer graphics is a classic forward modeling problem (given some objects, cameras, and lighting, simulate the images that would result), while computer vision problems are usually of the inverse kind (given one or more images, recover the scene that gave rise to these images).

Given an instance of an inverse problem, there are, in general, several ways to proceed. For instance, it may turn out that through clever (or sometimes straightforward) algebraic manipulation, a closed form solution for the unknowns can be derived. Consider, for example, the *camera matrix calibration* problem: given an image of a calibration pattern consisting of known 3D point positions, compute the 3×4 camera matrix \mathbf{P} that maps these points onto the image plane.

In more detail, we can write this problem as (see §[5.2.1](#)):

$$\begin{aligned} x_i &= \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}} \\ y_i &= \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}, \end{aligned} \tag{B.1}$$

where (x_i, y_i) is the feature position of the i th point measured in the image plane, (X_i, Y_i, Z_i) is the corresponding 3D point position, and the p_{ij} are the unknown entries of the camera matrix \mathbf{P} . Moving the denominator over to the left hand side, we end up with a set of simultaneous linear equations

$$\begin{aligned} x_i(p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}) &= p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03} \\ y_i(p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}) &= p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13} \end{aligned} \tag{B.2}$$

that we can solve to obtain an estimate of \mathbf{P} (more details on how to solve such a set of equations are given in §[A.2](#)).

The question then arises: is this set of equations the right ones to be solving? If the measurements are totally noise-free, or we do not care about getting the best possible answer, then the answer may very well be yes. However, in general, we cannot be sure that we have a reasonable algorithm unless we make a model of the likely sources of error, and devise an algorithm that performs as well as possible given these potential errors.

¹ In machine learning, these problems are called *regression problems*, because we are trying to estimate a *continuous* quantity from noisy inputs, as opposed to a discrete *discrimination* (classification) task ([Bishop 2006](#)).

B.1 Estimation theory

The study of such inference problems from noisy data is often called *estimation theory* [*Note: find one or more references: (Gelb 1974)*]. We first start by writing down the forward process that leads from our unknowns (and knowns) to a set of noise-corrupted measurements. Then, we devise an algorithm that will give us an estimate (or set of estimates) that are both insensitive to the noise (as best they can be), and also quantify the reliability of these estimates.

The specific equations given above (B.1) are just a particular instance of a more general set of *measurement equations*,

$$\mathbf{z}_i = \mathbf{f}_i(\mathbf{x}) + \mathbf{n}_i, \quad \mathbf{n}_i \sim N(0, \Sigma_i). \quad (\text{B.3})$$

Here, the \mathbf{z}_i are the noise-corrupted *measurements* (e.g., (x_i, y_i) in (B.1)), and \mathbf{x} is the unknown *state vector*.

Each measurement comes with its associated *measurement model* [*Note: check Gelb for terminology*] $\mathbf{f}_i(\mathbf{x})$ that maps the unknown into that particular measurement. An alternative formulation would be to have one general function $\mathbf{f}(\mathbf{x}, \mathbf{p}_i)$ that uses a per-measurement parameter vector \mathbf{p}_i that distinguishes between different measurements (e.g., (X_i, Y_i, Z_i) in (B.1)). Note that the use of the $\mathbf{f}_i(\mathbf{x})$ form makes it straightforward to have measurements of different dimensions, which will become useful when we start adding in prior information (§B.4).

Each measurement is also contaminated with some noise \mathbf{n}_i . In the above equation (B.3), we have indicated that \mathbf{n}_i is a zero-mean multi-dimensional normal (Gaussian) random variable with a covariance matrix Σ_i . In general, the noise need not be Gaussian, and in fact, it is usually prudent to assume that some measurements may be outliers. We will, however, defer this discussion to §B.3, until after we have explored the simpler Gaussian noise case more fully. We also assume that the noise vectors \mathbf{n}_i are independent. In the case where they are not (e.g., when some constant gain or offset contaminates all of the pixels in a given image), we can add this noise as a *nuisance parameter* to our state vector \mathbf{x} and later estimate its value (and discard it, if so desired).

Likelihood for multivariate Gaussian noise

Given all of the noisy measurements $\mathbf{z} = \{\mathbf{z}_i\}$, we would like to infer a probability distribution on the unknown \mathbf{x} vector. We can write the *likelihood* of having observed the $\{\mathbf{z}_i\}$ given a particular value of \mathbf{x} as

$$L = p(\mathbf{z}|\mathbf{x}) = \prod_i p(\mathbf{z}_i|\mathbf{x}) = \prod_i p(\mathbf{z}_i|\mathbf{f}_i(\mathbf{x})) = \prod_i p(\mathbf{n}_i). \quad (\text{B.4})$$

Since each noise vector \mathbf{n}_i is a multivariate Gaussian with covariance Σ_i , we can write this as

$$L = \prod_i |2\pi\Sigma_i|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{z}_i - \mathbf{f}_i(\mathbf{x}))^T \Sigma_i^{-1} (\mathbf{z}_i - \mathbf{f}_i(\mathbf{x}))\right) \quad (\text{B.5})$$

$$= \prod_i |2\pi\Sigma_i|^{-1/2} \exp\left(-\frac{1}{2}\|\mathbf{z}_i - \mathbf{f}_i(\mathbf{x})\|_{\Sigma_i^{-1}}^2\right),$$

where the matrix norm $\|\mathbf{x}\|_{\mathbf{A}}^2$ is a shorthand notation for $\mathbf{x}^T \mathbf{A} \mathbf{x}$. The norm $\|\mathbf{z}_i - \bar{\mathbf{z}}_i\|_{\Sigma_i^{-1}}$ is often called the *Mahalanobis distance* and is used to measure the distance between a measurement and the mean of a multivariate Gaussian distribution. Contours of equal Mahalanobis distance are equi-probability contours. Note that when the measurement covariance is isotropic (the same in all directions), i.e., when $\Sigma_i = \sigma_i^2 \mathbf{I}$, the likelihood can be written as

$$L = \prod_i (2\pi\sigma_i^2)^{-N_i/2} \exp\left(-\frac{1}{2\sigma_i^{2N_i}}\|\mathbf{z}_i - \mathbf{f}_i(\mathbf{x})\|^2\right), \quad (\text{B.6})$$

where N_i is the length of the i th measurement vector \mathbf{z}_i .

We can more easily visualize the structure of the covariance matrix and the corresponding Mahalanobis distance if we first perform an *eigenvalue* or *principal component* analysis (PCA) of the covariance matrix,

$$\Sigma = \Phi \operatorname{diag}(\lambda_0 \dots \lambda_{N-1}) \Phi^T. \quad (\text{B.7})$$

Equal-probability contours of the corresponding multi-variate Gaussian, which are also equi-distance contours in the Mahalanobis distance, are then multi-dimensional ellipsoids whose axis directions are given by the columns of Φ (the *eigenvectors*) and whose lengths are given by the $\sqrt{\lambda_i}$.

[Note: see (Hartley and Zisserman 2004) for more on Mahalanobis distance and notation]

It is usually more convenient to work with the negative log-likelihood, which we notate as a *cost* or *energy*

$$E = -\log L = \frac{1}{2} \sum_i (\mathbf{z}_i - \mathbf{f}_i(\mathbf{x}))^T \Sigma_i^{-1} (\mathbf{z}_i - \mathbf{f}_i(\mathbf{x})) + k \quad (\text{B.8})$$

$$= \frac{1}{2} \sum_i \|\mathbf{z}_i - \mathbf{f}_i(\mathbf{x})\|_{\Sigma_i^{-1}}^2 + k, \quad (\text{B.9})$$

where $k = -\sum_i \log |2\pi\Sigma_i|$ is a constant that depends on the measurement variances, but is independent of \mathbf{x} . Notice that the inverse covariance $\mathbf{C} = \Sigma_i^{-1}$ plays the role of a *weight* on each of the measurement error *residuals*, i.e., the difference between the contaminated measurement \mathbf{z}_i and its uncontaminated value $\mathbf{f}_i(\mathbf{x})$. In fact, the inverse covariance is often called the *information matrix* [Note: check if this is the Fischer (sp?) information matrix], since it tells us how much information is contained in a given measurement, i.e., how well it constrains the final estimate. We can also think of this matrix as associating the amount of *confidence* to associate with each measurement (hence the letter \mathbf{C}).

Note also that in this formulation, it is quite acceptable for some information matrices to be singular (of degenerate rank) or even zero (if the measurement is missing altogether). Rank-deficient

measurements often occur, for example, when using a line feature or edge to measure a 3D edge-like feature, since its exact position along the edge is unknown (of infinite or extremely large variance) (see §7.1.3).

In order to make the distinction between the noise contaminated measurement and its expected value for a particular setting of \mathbf{x} more explicit, we adopt the notation $\tilde{\mathbf{z}}$ for the former (think of the tilde as the approximate or noisy value), and $\hat{\mathbf{z}} = \mathbf{f}_i(\mathbf{x})$ for the latter (this of the hat as the predicted or expected value). [Note: Check that this is consistent with (Gelb 1974)]. We can then write the negative log-likelihood as

$$E = -\log L = \sum_i \|\tilde{\mathbf{z}}_i - \hat{\mathbf{z}}_i\|_{\Sigma_i^{-1}}. \quad (\text{B.10})$$

B.2 Maximum likelihood estimation and least squares

Now that we have developed the likelihood (and log-likelihood) function, how do we select a reasonable value for our state estimate \mathbf{x} . One plausible choice might be to choose the value of \mathbf{x} that maximizes $L = p(\mathbf{z}|\mathbf{x})$. In fact, in the absence of any prior model for \mathbf{x} (§B.4), we have

$$L = p(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}, \mathbf{x}) = p(\mathbf{x}|\mathbf{z}).$$

Therefore, choosing the value of \mathbf{x} that maximizes the likelihood is equivalent to choosing the maximum of our probability density estimate for \mathbf{x} .

When might this be a good idea? If the data (measurements) constrain the possible values of \mathbf{x} so that they all cluster tightly around one value (more specifically, if the distribution $p(\mathbf{x}|\mathbf{z})$ is a unimodal Gaussian), then the maximum likelihood estimate is the optimal one in that it is the estimate that is both unbiased and has the least possible variance. [Note: Review the probability/statistic literature to make this tighter]. In many other cases, if a single estimate is all that is required, it is still often the best estimate. However, if the probability is heavily multi-modal (lots of local minima in the log-likelihood), then much more care may be required. In particular, it might be necessary to defer certain decisions (such as the ultimate position of an object being tracked) until more measurements have been taken. §B.9 discusses one possible method for modeling and updating such multi-modal distributions.

[Note: Add a figure showing a uni-modal distribution vs. some multi-modal ones].

Another possible way to choose the best estimate is to maximize the *expected utility* (or conversely, to minimize the expected loss) associated with obtaining the correct estimate, i.e., by minimizing

$$E_{\text{loss}} = \int l(\mathbf{x} - \mathbf{y}) p(\mathbf{y}|\mathbf{z}) d\mathbf{y}. \quad (\text{B.11})$$

For example, if a robot wants to avoid hitting a wall at all costs, a scalar loss function can be high whenever the estimate underestimates the true distance to the wall. When $l(\mathbf{x} - \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y})$,

we obtain the maximum likelihood estimate, whereas when $l(\mathbf{x} - \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$, we obtain the least squares estimate.

How do we find the maximum likelihood estimate? If the measurement noise is Gaussian, we can minimize the quadratic objective function (B.10). This becomes even simpler if the measurement equations are linear, i.e.,

$$\mathbf{f}_i(\mathbf{x}) = \tilde{\mathbf{H}}_i \mathbf{x}. \quad (\text{B.12})$$

In this case, (B.10) becomes

$$E = \sum_i \|\tilde{\mathbf{z}}_i - \tilde{\mathbf{H}}_i \mathbf{x}\|_{\Sigma_i^{-1}} = \sum_i (\tilde{\mathbf{z}}_i - \tilde{\mathbf{H}}_i \mathbf{x})^T C_i (\tilde{\mathbf{z}}_i - \tilde{\mathbf{H}}_i \mathbf{x}), \quad (\text{B.13})$$

which is a simple quadratic form in \mathbf{x} , which can be directly minimized by solving an SPD system of linear equations §A.2.

When the measurements are non-linear, the system must be solved iteratively using non-linear least-squares §A.3.

B.3 Robust statistics

in general have outliers

talk about a *contaminated Gaussian*

robust statistics (Huber 1981, Black and Rangarajan 1996)

loss / penalty functions ($\log p(\mathbf{n}_i)$ not a quadratic)

mention *efficiency*, i.e., how well it averages away noise when there are no outliers (re-read books)

see (Triggs *et al.* 1999) for arguments about negative definite Hessian (also, look at what Sawnay does).

Talk about Median of Average Deviation (MAD) as a robust estimate of variance.

B.4 Prior models and Bayesian inference

(Szeliski 1990a)

regular priors

model selection and occam's razor (something by Torr)

marginalizing out unwanted (nuisance) variables: sometimes better results than MAP (MacKay 2003) [Note: is this what people call Gaussian or Dirichlet processes?]

B.5 Markov Random Fields

spatial priors: values similar to neighbors

correlated noise: doesn't fit in well with sparse system solving, can't deal with non-Gaussian priors

MRFs: simple local interactions (see Zhu for more complex filter-based)

also, do 1-D: snakes

Be sure to mention line processes (Geman and Geman 1984), and how they can usually be marginalized away (Black and Rangarajan 1996).

different solution techniques: Gibbs Sampler, graph cuts, mean field (Scharstein and Szeliski 1998), Swendsen-Wang (Barbu and Zhu 2003, Barbu and Zhu 2005) [Note: Mention that graph cuts is discussed in more detail in §4.5.4 Where does (Ishikawa 2003) fit in?]

More complex models: 3-D (Boykov and Funka-Lea 2006), N8 and larger pairwise neighborhoods (Rother QBPO (Rother *et al.* 2007)), higher order potentials (Kohli *et al.* 2008).

B.6 Inference algorithms

B.6.1 Gradient descent and simulated annealing

Continuation methods are useful in some cases...

Call simulated annealing “stochastic gradient descent”?

Also, ICM (Kittler and Föglein 1984, Besag 1986) [Note: read (Kittler and Föglein 1984) to make sure it's appropriate; it's cited in (Bishop 2006).]

HCF (Chou and Brown 1990)

B.6.2 Dynamic programming

Is it covered in (Bishop 2006)?

B.6.3 Belief propagation

Recent work:

Efficient Belief Propagation for Vision Using Linear Constraint Nodes, Brian Potetz, (Potetz 2007): How to do Belief Propagation (BP) when nodes have large degree but are structured as a non-linearity (or hard constraint) following a linear equation: just use variable substitution and partial integration (simple to understand, but significant breakthrough).

B.6.4 Graph cuts

Describe the basic binary graph formulation, originally developed by (Greig *et al.* 1989) and introduced to the computer vision community by (Boykov *et al.* 2001). Kolmogorov and Zabih (2004) formally characterize the class of binary energy potentials for which these results hold, while newer work by Komodakis *et al.* (2007) and Rother *et al.* (2007) provide good algorithms for the cases when they do not.

Show the min-cut / max-flow graph corresponding to the 1-D MRF.

Describe (Boykov *et al.* 2001) *swap move* and the *expansion move* in more detail, as they aren't covered in §3.6.2.

Recent work:

Fast, Approximately Optimal Solutions for Single and Dynamic MRFs, Nikos Komodakis, Georgios Tziritas, and Nikos Paragios, (Komodakis *et al.* 2007): Primal-dual algorithms can sometimes significantly outperform regular graph cuts. Should run these on our MRF study, and see if they help with Photomontage.

Optimizing Binary MRFs via Extended Roof Duality. Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. (Rother *et al.* 2007): Beautiful work showing how a "QBPO" algorithm can be used to "guess" the solution to a graph cut problem, usually outperforming all other approaches. Makes graph cuts practical for many non-submodular problems (like image stitching).

P^3 and Beyond: Solving Energies with Higher Order Cliques. Pushmeet Kohli, Pawan Mudigonda, and Philip Torr. (Kohli *et al.* 2007): Nice extension of solvable MRFs to an important class of higher-order cliques, with good results on video segmentation.

Graph Cut Based Optimization for MRFs with Truncated Convex Priors. Olga Veksler. (Veksler 2007): New MRF inference algorithm for truncated convex interaction potentials using Geiger's graph construction. Nice results on stereo and de-noising.

See if I've included all of these: (Yedidia *et al.* 2000, Felzenszwalb and Huttenlocher 2004b, Kohli and Torr 2005, Kumar and Hebert 2006, Kumar and Torr 2006, Szeliski *et al.* 2008c)

(Boykov and Funka-Lea 2006) also has a nice set of references—re-read. For example, w.r.t. efficient solvers, they state *Recently, (Boykov and Kolmogorov, 2004) studied the practical efficiency of combinatorial min-cut/maxflow algorithms on applications in computer vision. It was shown that some max-flow techniques could solve 2D and 3D segmentation problems in close to realtime using regular PCs. Further significant acceleration was demonstrated for dynamic segmentation problems using flow-recycling (Kohli and Torr, 2005) and cutrecycling (Juan and Boykov, 2006). Some versions of max-flow/min-cut algorithms can be run on parallel processors (Goldberg and Tarjan, 1988). Parallel implementations are also possible on Graphics Processing Units.*⁶ While straightforward implementation of graph cuts may require a lot of memory for 3D

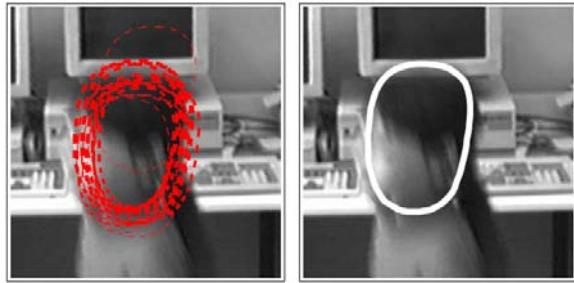


Figure B.1: A set of potential head locations (left) and their consensus estimate (right) (Isard and Blake 1998).

applications, recent results in Lombaert et al. (2005) showed that multi-level and banded techniques can alleviate the problem.

Also, More recently, (Boykov et al., 2006) showed a very strong connection between graph cuts and level-sets (Sethian, 1999; Osher and Fedkiw, 2002; Sapiro, 2001; Osher and Paragios, 2003). In particular, (Boykov et al., 2006) developed a novel integral approach to solving surface propagation PDEs based on combinatorial graph cuts algorithms. Such PDEs arise when computing gradient flow evolution of active contours which are very widely used in computer vision and medical image analysis. The results in Boykov et al. (2006) suggest that combinatorial graph cuts algorithms can be used as a robust numerical method for an important class of variational problems that was previously addressed mainly with level-sets. Be sure to cross-reference these relationships in level sets §4.4.3 and graph cut segmentation §4.5.4 sections.

[Note: Covered in the main part of the text?]

B.7 Uncertainty estimation (error analysis)

include eigenvalue analysis for dominant errors (Szeliski and Kang 1997)

Also, cite Förstner (2005), which discusses various ways to infer and model 3D lines in projective geometry, as well as how to estimate the uncertainty in such fitted models.

B.8 Kalman filtering

Bayesian and least squares techniques are a powerful way to draw inferences from noisy data and to reason about the uncertainties in our estimates. In many situations such as visual tracking, these inferences need to be repeated at every time interval. Unless each new frame is a completely independent instance of the inference problem, the previous estimate(s) of the system state can be a powerful predictor of the current state estimate.

Consider, for example, the task of tracking a rapidly moving head in a video sequence (Figure B.1). If the head is staying still and the measurements are noisy, we can average all of the (red) measurements to produce a consensus (white) estimate. Assuming that we know the covariance for each measurement, Σ_i , the final estimate can be computed as the minimum of (B.14), i.e.,

$$E = \sum_i \|\tilde{\mathbf{z}}_i - \tilde{\mathbf{H}}_i \mathbf{x}\|_{\Sigma_i^{-1}} = \sum_i (\tilde{\mathbf{z}}_i - \tilde{\mathbf{H}}_i \mathbf{x})^T \mathbf{C}_i (\tilde{\mathbf{z}}_i - \tilde{\mathbf{H}}_i \mathbf{x}). \quad (\text{B.14})$$

If the measurements directly measure the quantity of interest, i.e., $\tilde{\mathbf{H}}_i = \mathbf{I}$, this is simply a weighted average of the measurements,

$$\hat{\mathbf{x}} = \mathbf{C}^{-1} \sum_i \mathbf{C}_i \tilde{\mathbf{z}}_i, \quad (\text{B.15})$$

where $\mathbf{C} = \sum_i \mathbf{C}_i$ is the summed information matrix (inverse covariance).

incremental estimation: add measurements one at a time, increment Hessian/information matrix, re-solve (rank-one updates)

Welch's Kalman filter page: <http://www.cs.unc.edu/~welch/kalman/> also includes their TR (Welch and Bishop 1995), which references (Gelb 1974, Sorenson 1980, Maybeck 1982) and others [Grewal93; Lewis86; Brown92; Jacobs93]. [Note: Re-read (Gelb 1974, Maybeck 1982).]

[Note: derive and present the inverse covariance (information filter) formulation, as explained in (Szeliski 1989, Szeliski 1990a)]

Use the notation

$$\mathbf{x}_t = \mathbf{A} \mathbf{x}_{t-1} + \mathbf{w} \quad (\text{B.16})$$

Or, should I reserve \mathbf{A} for the information matrix, and use something like \mathbf{F} for the process model?

process model, in addition to measurement model

non-linear (extended and linearized Kalman filters)

moving lag, Kalman smoothers: mention, and cite (Gelb 1974) (also (Maybeck 1982) for computational issues)

Applications: recursive s.f.m. (Azarbayejani and Pentland 1995, McLauchlan and Murray 1995, McLauchlan 2000), recursive stereo (Matthies *et al.* 1989), recursive snakes (Terzopoulos and Szeliski 1992)

Nice review of relationships between Kalman filters, PCA, ICA, FA, etc by Roweis and Ghahramani (1999).

What's in this article (Jordan *et al.* 1999)?

B.9 Particle filtering

[Note: Re-use some of the figures in the CONDENSATION section, and give the formulas from (Isard and Blake 1998).]

Appendix C

Supplementary material

C.1	Data sets	532
C.2	Software	532
	C.2.1 GPU implementation	534
C.3	Slides	537
C.4	Bibliography	538

C.1 Data sets

Provide images with answers (3D reconstructions, features, stitches, etc.)

Useful test data sets (some with solutions) are also provided so that students can test and evaluate their algorithms without the need for acquiring their own photographic material. (Most students, of course, will probably want to try their newly developed algorithms on data they acquire themselves :-) [*Note: For exercises, provide one data set with a known solution, and one or more data sets without the answer?*]

To do this, first re-organize all your figures and/or images by chapter (or is this part unnecessary)? Then, write a SED script to convert you .tex chapter files into figures only .tex files. Convert these with Tex2HTML, and then put the HTML files into the appropriate images directory. A top-level index file will then give the image to figure association. For Visio / Excel figures, either include these, or export PDFs to some other format (.PNG). Compare lossless JPG to PNG for compression efficiency (where alpha channel is unneeded).

Also, pointers to more data sets on the Web: Computer Vision Home Page; Middlebury Stereo, MView, MRF, and Flow pages; LabelMe; Berkeley Segmentation database; BRDF / BTF database at Columbia.

Be sure to mention the ([Martin *et al.* 2001](#)) database, as well as the newer ([Unnikrishnan *et al.* 2007](#)) evaluation.

The database of foreground/background segmentations used in ([Alpert *et al.* 2007](#)) is at http://www.wisdom.weizmann.ac.il/~vision/Seg_Evaluation_DB/index.html

Plus, photo sharing sites with labels: Flickr, SmugMug

Personal photos or experimental datasets ICCV'2005 datasets: Bellevue, Stata, Quincy Market, Haight;

Photo Tourism datasets: Prague, Great Wall, maybe cabin fisheyes

Panoramas: Lake Wenatchee, some from Europe

Old turntable sequences

Personal: Kids HDR, Monet Iris, ...

!

C.2 Software

Drop the idea of writing my own package, but give pointers to OpenCV, NASA package, ...

Check out Intel's OpenCV library ([Bradsky and Kaehler 2008](#)) and see what algorithms are provided there. Also, there's some new library at NASA Ames, there's Vantage and Phil Torr's MatLab code...

Also, give a list of popular image editing packages, so that students can compare their results against those: Windows Live Photo Gallery, Microsoft Office Picture Manager, iPhoto, Picassa, JASC, irfanView, ... , Photoshop

Other CV packages: Greek bundle adjuster...

[Note: Fredo's lecture notes contain tons of pointers: incorporate a subset of these (also have pointers to images, e.g., HDR).]

Software by chapter in the book

Segmentation by weighted aggregation (SWA) (Alpert *et al.* 2007), Windows implementation at www.cs.weizmann.ac.il/~vision/SWA/.

§4.5.2

EDISON software for intensity-based mean-shift segmentation at <http://www.caip.rutgers.edu/riul/research/robust.html>

sample test images at http://www.caip.rutgers.edu/~comanici/segm_images.html

§4.5.3

Normalized cuts segmentation including intervening contours (Shi and Malik 2000, Malik *et al.* 2001), Matlab implementation at www.cis.upenn.edu/~jshi/.

Gaussian noise generation. A lot of basic software packages come with a uniform random noise generator (e.g., the `rand()` routine in Unix), but not all have a Gaussian random noise generator. To compute a normally distributed random variable of variance σ^2 , note that the two-dimensional Gaussian distribution [Note: fill this in ...] The C code is given below:

```
double urand()
{
    return ((double) rand()) / ((double) RAND_MAX);
}

void grand(double& g1, double& g2)
{
#ifndef M_PI
#define M_PI      3.14159265358979323846
#endif // M_PI

    double n1 = urand();
    double n2 = urand();
    double sqlogn1 = sqrt(-2.0 * log (n1));
```

```

double angl = (2.0 * M_PI) * n2;
g1 = sqlogn1 * cos(angl);
g2 = sqlogn1 * sin(angl);
}

```

Pseudocolor generation. In many applications, it is convenient to be able to visualize the set of labels assigned to an image (or to image features such as lines). One of the easiest ways to do this is to assign a unique color to each integer label. In my work, I have found it convenient to distribute these labels in a quasi-uniform fashion around the RGB color cube using the following idea.

For each (non-negative) label value, consider the bits as being split among the three color channels, e.g., for a 9-bit value, the bits could be labeled RGBRGBRGB. After collecting each of the three color value, *reverse* the bits so that the low-order bits vary the quickest. In practice, for 8-bit color channels, this bit reverse can be stored in a table, or a complete table mapping from labels to pseudocolors (say with 4K entries) can be pre-computed.

C.2.1 GPU implementation

Talk about how pixel shaders combined with texture-mapped rendering can be used to implement point processes, neighborhood operations, and geometric warping. Less obvious how to do Fourier transforms and optimization, but GPUs have successfully been used (see GPGPU.org). Also, newest architectures such as CUDA make it possible to write more general algorithms. multicore (with multimedia acceleration) is another compelling image processing architecture. Similar to late 80's when data-parallel architectures such as the Connection Machine and MasPar had their heyday.

[*Note: See hand-drawn sketches from Sept 8, 2007*]

Algorithms in later part of the book, such as feature detection, sparse system solving, flow and stereo matching, recognition, and especially computational photography, image-based rendering, and video processing, are all amenable to greater or lesser extents to GPU acceleration.

(Reprise this theme in the conclusions section.)

[*Note: The rest of this chapter/section was old notes on how to synchronize the textbook with a proposed software package...*]

Redo to match chapters and software structure

Can use a variety of systems to implement algorithms in this book

MatLab: has image and matrix classes built in, image resampling

C++: need to write image and matrix classes, link to LaPack++ for numerics, SparseLib or SparseKit (?) for Sparse matrix (MatLab already has this built in).

C#: need similar classes, some support for graphics (Display.Net)

For graphics: OpenGL vs. Direct3D (or something better?)

Alternative: use a VRML viewer: easy to write, interactive viewer

[Note: Just because an algorithm is described in this book doesn't mean you have the right to use it commercially. For commercial use, it's your responsibility to license any intellectual property you may be using...]

Coordinates and transformations

Images are (sampled) functions from $\Re^2 \rightarrow \Re$, but since only a subset of the domain is sampled, we need to know the origin (and its spacing). A 3×3 matrix would suffice for this. (Note that this is the sense of the transforms in VideoMosaic, i.e., they specify mappings from image coordinates to a virtual compositing plane. This is the opposite of what camera matrices usually do.)

Perspective projections using 4×4 matrices

We use 4-vectors and 4×4 matrices in the software library to minimize the amount of clutter (duplicated functionality). If you care about ultimate efficiency or preciseness of notation, go ahead and implement your own 3- and 2-dimensional versions.

[Note: Provide definitions for 2- and 3- vectors and matrices for consistency of notation, but don't define any functions, unless we need them later on.]

Float vs. double: double is usually safer (e.g., inferring precise rotation angles for a mosaic may require this much precision: verify this, and if so, publish it). If you want smaller memory footprint and/or plan to use a GPU or CPU SIMD instructions, may want to use single-precision floats.

Use typedefs in this library to cover `vec4<float>` with `vec4f` and `mat4x4<float>` with `mat4x4f` (or should it be `float4` and `float4x4`?). Similarly, cover `vec4<double>` and `mat4x4<double>`. We could typedef `vec4d` and `mat4x4d` with `vec4` and `mat4x4`, but I think this would just be too confusing.

Provide type casts from float-double and vice versa. Can do this using templates (with two types and casts).

Need to specify dimensionality on common vector operations: Dot2, Dot3, Dot4, MagSqN, MagN, Norm, AddN, SubN, ScaleN In most cases (where vector is returned), need some policy on “un-used” coordinates.

Radial distortion?

Co-vectors?

Persistence/serialization and parameters How to persist sets of coordinates, lines? Use curly brace notation (MatLab interoperability)? XML? Tabular form with headings (Excel interoperability)?

What about parameters?

Vectors and matrices

Heap allocation, (smart) pointers, and garbage collection (reference counting).

Vectors: shallow copy, support `push_back` (since so useful)

Matrices: same smart pointer, but 2D access.

Same discussion on precision. Here, concerned with arbitrary size.

[Note: Echo same structure as Appendix A]

Basic linear algebra (matrix multiplies).

Matrix operations: QR, eigenvalue, SVD, Cholesky.

Recommend linking to LAPack++, but maybe provide some simple implementations?

Least squares

Sparse matrix representation (linear equations): index/value pairs.

(Symmetric) sparse matrix representation: skyline

Sparse matrix techniques:

- direct (Cholesky)
- iterative (conjugate gradient)

Non-linear regression (optimization)

Provide hooks for creating linearized equations.

Call sparse LDU

Track the error, increase lambda if necessary. Or, take a smaller step.

Images and image processing

Image storage class (like `matrix - mat[T_i]`), but has a third dimension (band). Also, has a name slot.

Have a pose slot, or create a hybrid structure (say `camera = image + pose`)?

Point transforms (convolution, pyramids (Burt and Adelson 1983a), morphology, compositing (Porter and Duff 1984, Blinn 1994a))

Global transforms (rotations, resampling) (Heckbert 1986, Wolberg 1990)

Local warps

Calibration, pose estimation, and structure from motion

Image alignment

Stereo correspondence

Volumetric representations

Surface representations

Visualization / display / rendering / graphics

Need some drawing package to overlay feature locations on images. Ideally, we can have one set of high-level commands that emit PostScript, rendered images, SVG (for embedding in PPT?), ... Structure it like gl_spoof? Full 3D matrix, so we can visualize 3D reconstruction overlaid on input images. This package is also useful for generating synthetic test data.

Need to implement clipping in software if driving a simple 2D package like PostScript or SVG.

Have a script language to generate the figures/data? This might be convenient.

Is it worth re-generating *all* of the line drawing figures in this paper with such a package?
Probably not.

C.3 Slides

Provide PowerPoint lectures to go with each chapter, using the Visio drawings and TeXPoint for formulas.

(Black on white works easiest to transfer drawings and equations.)

[Note: How to include videos? Number them just like Figures. In an HTML version of the book (?) or PDF (???), can embed videos. We could consider co-publishing as an E-book, whatever that means. Also, how were formulas included in the SIGGRAPH 2004 papers 0297 HTML submission that has embedded videos (see PDFs/SG04_297CD/)? Did they use LaTeX2HTML? The HTML gives no clue...]

[Note: How about figures? If some are generated by my visualization software, it would be nice to have them embeddable in PPT, as well as EPS. What about generating interactive figures? Leonard McMillan does this on some of his Web page slides using Java, but probably too much work.]

C.4 Bibliography

Provide the bibliography in BibTeX and HTML formats, with links to all on-line papers.

Production notes Notes on using Acrobat: to remove whitespace around converted figures, use Document — Crop Pages (Ctrl+Shft+T), when Remove White Margins.

To crop a figure out of an existing PDF, use Tools — Advanced Editing — Crop Tool and then double click inside region.

If the document is in PDF v1.6, which pdflatex won't recognize, use Advanced — PDF Optimizer and choose Acrobat 5.0 or later, which generates PDF v1.4.

[Note: Is it preferable to list all authors, or to use et al. for more than two? Also, use full first names or just initials?]

Bibliography

- Van Huffel, S. and Vandewalle, J. (1991). *The Total Least Squares Problem: Computational Aspects and Analysis*. Society for Industrial and Applied Mathematics, Philadelphia.
- Abdel-Hakim, A. and Farag, A. (2006). CSIFT: A SIFT descriptor with color invariant characteristics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 1978–1983, New York City, NY.
- Adelson, E. H. and Bergen, J. (1991). The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20, MIT Press, Cambridge, MA.
- Adelson, E. H., Simoncelli, E., and Hingorani, R. (1987). Orthogonal pyramid transforms for image coding. In *SPIE Vol. 845 Visual Communications and Image Processing II*, pages 50–58, Society of Photo-Optical Instrumentation Engineers, Cambridge, Massachusetts.
- Agarwala, A. (2007). Efficient gradient-domain compositing using quadtrees. *ACM Transactions on Graphics*, 26(3).
- Agarwala, A., Hertzmann, A., Seitz, S., and Salesin, D. (2004). Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics*, 23(3), 584–591.
- Agarwala, A., Agrawala, M., Cohen, M., Salesin, D., and Szeliski, R. (2006). Photographing long scenes with multi-viewpoint panoramas. *ACM Transactions on Graphics*, 25(3), 853–861.
- Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D. H., and Cohen, M. F. (2004). Interactive digital photomontage. *ACM Transactions on Graphics*, 23(3), 292–300.
- Agarwala, A., Zheng, K. C., Pal, C., Agrawala, M., Cohen, M., Curless, B., Salesin, D., and Szeliski, R. (2005). Panoramic video textures. *ACM Transactions on Graphics*, 24(3), 821–827.
- Agin, G. J. and Binford, T. O. (1976). Computer description of curved objects. *IEEE Transactions on Computers*, C-25(4), 439–449.
- Akenine-Möller, T. and Haines, E. (2002). *Real-Time Rendering*. A K Peters, Wellesley, Massachusetts, second edition.

- Aliaga, D. G. *et al.*. (2003). Sea of images. *IEEE Computer Graphics and Applications*, 23(6), 22–30.
- Aloimonos, J. (1990). Perspective approximations. *Image and Vision Computing*, 8, 177–192.
- Alpert, S., Galun, M., Basri, R., , and Brandt, A. (2007). Image segmentation by probabilistic bottom-up aggregation and cue integration. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Ameller, M.-A., Triggs, B., and Quan, L. (2000). Camera pose revisited – new linear algorithms. <http://www.inrialpes.fr/movi/people/Triggs/home.html>.
- Amini, A. A., Weymouth, T. E., and Jain, R. C. (1990). Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9), 855–867.
- Anandan, P. (1984). Computing dense displacement fields with confidence measures in scenes containing occlusion. In *Image Understanding Workshop*, pages 236–246, Science Applications International Corporation, New Orleans.
- Anandan, P. (1989). A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3), 283–310.
- Anandan, P. and Irani, M. (2002). Factorization with uncertainty. *International Journal of Computer Vision*, 49(2-3), 101–116.
- Andriluka, M., Roth, S., and Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Antone, M. and Teller, S. (2000). Recovering relative camera rotations in urban scenes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 282–289, Hilton Head Island.
- Argyriou, V. and Vlachos, T. (2003). Estimation of sub-pixel motion using gradient cross-correlation. *Electronic Letters*, 39(13), 980–982.
- Arnold, R. D. (1983). *Automated Stereo Perception*. Technical Report AIM-351, Artificial Intelligence Laboratory, Stanford University.
- Arya, S. *et al.*. (1998). An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6), 891–923.
- Ashdown, I. (1993). Near-field photometry: A new approach. *Journal of the Illuminating Engineering Society*, 22(1), 163–180.

- Atkinson, K. B. (1996). *Close Range Photogrammetry and Machine Vision*. Whittles Publishing, Scotland, UK.
- Aurich, V. and Weule, J. (1995). Non-linear gaussian filters performing edge preserving diffusion. In *17th DAGM-Symposium*, pages 538–545, Bielefeld.
- Avidan, S. (2001). Support vector tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 283–290, Kauai, Hawaii.
- Ayache, N. (1989). *Vision Stéréoscopique et Perception Multisensorielle*. InterEditions., Paris.
- Azarbeyjani, A. and Pentland, A. P. (1995). Recursive estimation of motion, structure, and focal length. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6), 562–575.
- Bab-Hadiashar, A. and Suter, D. (1998a). Robust optic flow computation. *International Journal of Computer Vision*, 29(1), 59–77.
- Bab-Hadiashar, A. and Suter, D. (1998b). Robust total least squares based optic flow computation. In *Asian Conference on Computer Vision (ACCV'98)*, pages 566–573, ACM, Hong Kong.
- Badra, F., Qumsieh, A., and Dudek, G. (1998). Rotation and zooming in image mosaicing. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 50–55, IEEE Computer Society, Princeton.
- Bae, S., Paris, S., and Durand, F. (2006). Two-scale tone management for photographic look. *ACM Transactions on Graphics*, 25(3), 637–645.
- Bai, X. and Sapiro, G. (2007). A geodesic framework for fast interactive image and video segmentation and matting. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Baker, H. H. (1977). Three-dimensional modeling. In *Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 649–655.
- Baker, H. H. (1982). *Depth from Edge and Intensity Based Stereo*. Technical Report AIM-347, Artificial Intelligence Laboratory, Stanford University.
- Baker, H. H. (1989). Building surfaces of evolution: The weaving wall. *International Journal of Computer Vision*, 3(1), 50–71.
- Baker, H. H. and Binford, T. O. (1981). Depth from edge and intensity based stereo. In *IJCAI81*, pages 631–636.
- Baker, H. H. and Bolles, R. C. (1989). Generalizing epipolar-plane image analysis on the spatiotemporal surface. *International Journal of Computer Vision*, 3(1), 33–49.
- Baker, S. and Kanade, T. (2002). Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9), 1167–1183.

- Baker, S. and Matthews, I. (2004). Lucas-Kanade 20 years on: A unifying framework: Part 1: The quantity approximated, the warp update rule, and the gradient descent approximation. *International Journal of Computer Vision*, 56(3), 221–255.
- Baker, S. and Nayar, S. (1999). A theory of single-viewpoint catadioptric image formation. *International Journal of Computer Vision*, 5(2), 175–196.
- Baker, S. and Nayar, S. K. (2001). Single viewpoint catadioptric cameras. In Benosman, R. and Kang, S. B., editors, *Panoramic Vision: Sensors, Theory, and Applications*, pages 39–71, Springer, New York.
- Baker, S. *et al.*. (2003a). *Lucas-Kanade 20 Years On: A Unifying Framework: Part 2*. Technical Report CMU-RI-TR-03-01, The Robotics Institute, Carnegie Mellon University.
- Baker, S. *et al.*. (2003b). *Lucas-Kanade 20 Years On: A Unifying Framework: Part 3*. Technical Report CMU-RI-TR-03-35, The Robotics Institute, Carnegie Mellon University.
- Baker, S. *et al.*. (2004). *Lucas-Kanade 20 Years On: A Unifying Framework: Part 4*. Technical Report CMU-RI-TR-04-14, The Robotics Institute, Carnegie Mellon University.
- Baker, S., Szeliski, R., and Anandan, P. (1998). A layered approach to stereo reconstruction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 434–441, Santa Barbara.
- Baker, S., Black, M., Lewis, J. P., Roth, S., Scharstein, D., and Szeliski, R. (2007). A database and evaluation methodology for optical flow. In *Eleventh International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brazil.
- Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary patterns. *Pattern Recognition*, 13(2), 111–122.
- Ballard, D. H. and Brown, C. M. (1982). *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Bar-Joseph, Z. *et al.*. (2001). Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), 120–135.
- Bar-Shalom, Y. and Fortmann, T. E. (1988). *Tracking and data association*. Academic Press, Boston.
- Barash, D. (2002). A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6), 844–847.
- Barbu, A. and Zhu, S.-C. (2003). Graph partition by Swendsen-Wang cuts. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 320–327, Nice, France.
- Barbu, A. and Zhu, S.-C. (2005). Generalizing Swendsen-Wang to sampling arbitrary posterior probabilities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9), 1239–1253.

- Barkans, A. C. (1997). High quality rendering using the Talisman architecture. In *Proceedings of the Eurographics Workshop on Graphics Hardware*.
- Barnard, S. T. (1989). Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1), 17–32.
- Barnard, S. T. and Fischler, M. A. (1982). Computational stereo. *Computing Surveys*, 14(4), 553–572.
- Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1), 43–77.
- Barrow, H. G. and Tenenbaum, J. M. (1981). Computational vision. *Proceedings of the IEEE*, 69(5), 572–595.
- Bartels, R. H., Beatty, J. C., and Barsky, B. A. (1987). *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers, Los Altos.
- Bartoli, A. (2003). Towards gauge invariant bundle adjustment: A solution based on gauge dependent damping. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 760–765, Nice, France.
- Bartoli, A. and Sturm, P. (2003). Multiple-view structure and motion from line correspondences. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 207–212, Nice, France.
- Bartoli, A., Coquerelle, M., and Sturm, P. (2004). A framework for pencil-of-points structure-from-motion. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 28–40, Springer-Verlag, Prague.
- Bascle, B., Blake, A., and Zisserman, A. (1996). Motion deblurring and super-resolution from an image sequence. In *Fourth European Conference on Computer Vision (ECCV'96)*, pages 573–582, Springer-Verlag, Cambridge, England.
- Bathe, K.-J. (2007). *Finite Element Procedures*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
- Baudisch, P., Tan, D., Steedly, D., Rudolph, E., Uyttendaele, M., Pal, C., and Szeliski, R. (2005). Panoramic viewfinder: providing a real-time preview to help users avoid flaws in panoramic pictures. In *OZCHI 2005*, Canberra, Australia.
- Baumberg, A. (2000). Reliable feature matching across widely separated views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 774–781, Hilton Head Island.
- Baumgart, B. G. (1974). *Geometric Modeling for Computer Vision*. Technical Report AIM-249, Artificial Intelligence Laboratory, Stanford University.

- Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 404–417, Springer.
- Beardsley, P., Torr, P., and Zisserman, A. (1996). 3D model acquisition from extended image sequences. In *Fourth European Conference on Computer Vision (ECCV'96)*, pages 683–695, Springer-Verlag, Cambridge, England.
- Beare, R. (2006). A locally constrained watershed transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7), 1063–1074.
- Becker, S. and Bove, V. M. (1995). Semiautomatic 3-D model extraction from uncalibrated 2-d camera views. In *SPIE Vol. 2410, Visual Data Exploration and Analysis II*, pages 447–461, Society of Photo-Optical Instrumentation Engineers, San Jose.
- Beier, T. and Neely, S. (1992). Feature-based image metamorphosis. *Computer Graphics (SIGGRAPH'92)*, 26(2), 35–42.
- Beis, J. S. and Lowe, D. G. (1999). Indexing without invariants in 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10), 1000–1015.
- Belhumeur, P. N. (1996). A Bayesian approach to binocular stereopsis. *International Journal of Computer Vision*, 19(3), 237–260.
- Belhumeur, P. N., Hespanha, J. P., and Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), 711–720.
- Belongie, S. and Malik, J. (1998). Finding boundaries in natural images: a new method using point descriptors and area completion. In *Fifth European Conference on Computer Vision (ECCV'98)*, pages 751–766, Springer-Verlag, Freiburg, Germany.
- Belongie, S. *et al.*. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4), 509–522.
- Bennett, E., Uyttendaele, M., Zitnick, L., Szeliski, R., and Kang, S. B. (2006). Video and image Bayesian demosaicing with a two color image prior. In *Ninth European Conference on Computer Vision (ECCV 2006)*, pages 508–521, Springer-Verlag, Graz.
- Benosman, R. and Kang, S. B., editors. (2001). *Panoramic Vision: Sensors, Theory, and Applications*, Springer, New York.
- Bergen, J. R., Anandan, P., Hanna, K. J., and Hingorani, R. (1992). Hierarchical model-based motion estimation. In *Second European Conference on Computer Vision (ECCV'92)*, pages 237–252, Springer-Verlag, Santa Margherita Liguere, Italy.

- Bertalmio, M. *et al.*. (2000). Image inpainting. In *Proceedings of ACM SIGGRAPH 2000*, pages 417–424.
- Bertalmio, M. *et al.*. (2003). Simultaneous structure and texture image inpainting. *IEEE Transactions on Image Processing*, 12(8), 882–889.
- Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society B*, 48(3), 259–302.
- Besl, P. (1989). Active optical range imaging sensors. In Sanz, J. L., editor, *Advances in Machine Vision*, chapter 1, pages 1–63, Springer-Verlag.
- Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 239–256.
- Besl, P. J. and Jain, R. C. (1985). Three-dimensional object recognition. *Computing Surveys*, 17(1), 75–145.
- Betrisey, C. *et al.*. (2000). Displaced filtering for patterned displays. In *Society for Information Display Symposium*, pages 296–299.
- Beymer, D. (1996). Feature correspondence by interleaving shape and texture computations. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 921–928, San Francisco.
- Bimber, O. (2006). Computational photography—the next big step. *Computer*, 39(8), 28–29.
- Birchfield, S. and Tomasi, C. (1998a). Depth discontinuities by pixel-to-pixel stereo. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 1073–1080, Bombay.
- Birchfield, S. and Tomasi, C. (1998b). A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4), 401–406.
- Birchfield, S. and Tomasi, C. (1999a). Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3), 269–293.
- Birchfield, S. and Tomasi, C. (1999b). Multiway cut for stereo and motion with slanted surfaces. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 489–495, Kerkyra, Greece.
- Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, New York, NY.
- Bitouk, D., Kumar, N., Dhillon, S., Belhumeur, P., and Nayar, S. K. (2008). Face swapping: Automatically replacing faces in photographs. *ACM Transactions on Graphics*, 27(3).

- Black, M., Yacoob, Y., Jepson, A. D., and Fleet, D. J. (1997). Learning parameterized models of image motion. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 561–567, San Juan, Puerto Rico.
- Black, M. J. and Anandan, P. (1996). The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1), 75–104.
- Black, M. J. and Jepson, A. D. (1998). EigenTracking: robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision*, 26(1), 63–84.
- Black, M. J. and Rangarajan, A. (1996). On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19(1), 57–91.
- Black, M. J., Sapiro, G., Marimont, D. H., and Heeger, D. (1998). Robust anisotropic diffusion. *IEEE Transactions on Image Processing*, 7(3), 421–432.
- Blake, A. and Isard, M. (1998). *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer Verlag, London.
- Blake, A. et al.. (2004). Interactive image segmentation using an adaptive GMMRF model. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 428–441, Springer-Verlag, Prague.
- Blake, A. and Zisserman, A. (1987). *Visual Reconstruction*. MIT Press, Cambridge, Massachusetts.
- Blake, A., Curwen, R., and Zisserman, A. (1993). A framework for spatio-temporal control in the tracking of visual contour. *International Journal of Computer Vision*, 11(2), 127–145.
- Blake, A., Zimmerman, A., and Knowles, G. (1985). Surface descriptions from stereo and shading. *Image and Vision Computing*, 3(4), 183–191.
- Blanz, V. and Vetter, T. (August 1999). A morphable model for the synthesis of 3d faces. *Proceedings of SIGGRAPH 99*, , 187–194. ISBN 0-20148-560-5. Held in Los Angeles, California.
- Blinn, J. (1998). *Dirty Pixels*. Morgan Kaufmann Publishers, San Francisco.
- Blinn, J. F. (1994a). Jim Blinn's corner: Compositing, part 1: Theory. *IEEE Computer Graphics and Applications*, 14(5), 83–87.
- Blinn, J. F. (1994b). Jim Blinn's corner: Compositing, part 2: Practice. *IEEE Computer Graphics and Applications*, 14(6), 78–82.
- Blinn, J. F. (1999). Jim Blinn's corner: A ghost in a snowstorm. *IEEE Computer Graphics and Applications*, 18(1), 79–84.
- Blinn, J. F. (2003). *Jim Blinn's Corner: Notation, Notation, Notation*. Morgan Kaufmann, San Francisco.

- Blinn, J. F. and Newell, M. E. (1976). Texture and reflection in computer generated images. *Communications of the ACM*, 19(10), 542–547.
- Bobick, A. F. and Intille, S. S. (1999). Large occlusion stereo. *International Journal of Computer Vision*, 33(3), 181–200.
- Boden, M. A. (2006). *Mind As Machine: A History of Cognitive Science*. Oxford University Press, Oxford, England.
- Bogart, R. G. (1991). View correlation. In Arvo, J., editor, *Graphics Gems II*, pages 181–190, Academic Press, Boston.
- Boiman, O., Shechtman, E., and Irani, M. (2008). In defense of nearest-neighbor based image classification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Boissonat, J.-D. (1984). Representing 2D and 3D shapes with the Delaunay triangulation. In *Seventh International Conference on Pattern Recognition (ICPR'84)*, pages 745–748, Montreal, Canada.
- Bolles, R. C., Baker, H. H., and Hannah, M. J. (1993). The JISCT stereo evaluation. In *Image Understanding Workshop*, pages 263–274, Morgan Kaufmann Publishers.
- Bolles, R. C., Baker, H. H., and Marimont, D. H. (1987). Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1, 7–55.
- Borgefors, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34(3), 227–248.
- Bosch, A., Varma, M., and Zisserman, A. (2008). Object detection using multiple kernel learning. In *British Machine Vision Conference (BMVC 2008)*, Springer-Verlag, Leeds.
- Bouguet, J.-Y. and Perona, P. (1998). 3d photography on your desk. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 43–50, Bombay.
- Boult, T. E. and Kender, J. R. (1986). Visual surface reconstruction using sparse depth data. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'86)*, pages 68–76, IEEE Computer Society Press, Miami Beach.
- Bovik, A., editor. (2000). *Handbook of Image and Video Processing*, Academic Press, San Diego.
- Bowyer, K. W., Kranenburg, C., and Dougherty, S. (2001). Edge detector evaluation using empirical roc curves. *Computer Vision and Image Understanding*, 84(1), 77–103.
- Boyer, E. and Berger, M. O. (1997). 3D surface reconstruction using occluding countours. *International Journal of Computer Vision*, 22(3), 219–233.

- Boykov, Y. and Funka-Lea, G. (2006). Graph cuts and efficient N-D image segmentation. *International Journal of Computer Vision*, 70(2), 109–131.
- Boykov, Y. and Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 105–112, Vancouver, Canada.
- Boykov, Y. and Kolmogorov, V. (2001). An experimental comparison of min-cut/max-flow algorithms for energy minimization in computer vision. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 205–220.
- Boykov, Y. and Kolmogorov, V. (2003). Computing geodesics and minimal surfaces via graph cuts. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 26–33, Nice, France.
- Boykov, Y., Veksler, O., and Zabih, R. (1998). A variable window approach to early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12), 1283–1294.
- Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11), 1222–1239.
- Bracewell, R. N. (1986). *The Fourier Transform and its Applications*. McGraw-Hill, New York, 2nd edition.
- Bradsky, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly.
- Brandt, A. (1986). Algebraic multigrid theory: The symmetric case. *Applied Mathematics and Computation*, 19(1-4), 23–56.
- Bregler, C. and Malik, J. (1998). Tracking people with twists and exponential maps. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 8–15, Santa Barbara.
- Bregler, C., Covell, M., and Slaney, M. (1997). Video rewrite: Driving visual speech with audio. *Computer Graphics (SIGGRAPH'97)*, , 353–360.
- Brice, C. R. and Fennema, C. L. (1970). Scene analysis using regions. *Artificial Intelligence*, 1(3-4), 205–226.
- Briggs, W. L. (1987). *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia.
- Briggs, W. L., Henson, V. E., and McCormick, S. F. (2000). *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, second edition.
- Brillaut-O'Mahoney, B. (1991). New method for vanishing point detection. *Computer Vision, Graphics, and Image Processing*, 54(2), 289–300.

- Brinkmann, R. (2008). *The Art and Science of Digital Compositing*. Morgan Kaufmann Publishers, San Francisco, 2nd edition.
- Broadhurst, A., Drummond, T. W., and Cipolla, R. (2001). A probabilistic framework for space carving. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 388–393, Vancouver, Canada.
- Brown, D. C. (1971). Close-range camera calibration. *Photogrammetric Engineering*, 37(8), 855–866.
- Brown, L. G. (1992). A survey of image registration techniques. *Computing Surveys*, 24(4), 325–376.
- Brown, M. and Lowe, D. (2002). Invariant features from interest point groups. In *British Machine Vision Conference*, pages 656–665, Cardiff, Wales.
- Brown, M. and Lowe, D. (2003a). Recognising panoramas. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1218–1225, Nice, France.
- Brown, M. and Lowe, D. (2003b). Unsupervised 3D object recognition and reconstruction in unordered datasets. In *International Conference on 3D Imaging and Modelling*, pages 1218–1225, Nice, France.
- Brown, M. and Lowe, D. (2007). Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1), 59–73.
- Brown, M., Szeliski, R., and Winder, S. (2004). *Multi-Image Matching Using Multi-Scale Oriented Patches*. Technical Report MSR-TR-2004-133, Microsoft Research.
- Brown, M., Szeliski, R., and Winder, S. (2005). Multi-image matching using multi-scale oriented patches. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 510–517, San Diego, CA.
- Brown, M., Hartley, R., , and Nistér, D. (2007). Minimal solutions for panoramic stitching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Brown, M. Z., Burschka, D., and Hager, G. D. (2003). Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8), 993–1008.
- Brox, T. *et al.*. (2004). High accuracy optical flow estimation based on a theory for warping. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 25–36, Springer-Verlag, Prague.
- Brubaker, S. C. *et al.*. (2008). On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, 77(1-3), 65–86.
- Bruhn, A., Weickert, J., and Schnorr, C. (2005). Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods. *International Journal of Computer Vision*, 61(3), 211–231.

- Buades, A., Coll, B., and Morel, J.-M. (2005). A non-local algorithm for image denoising. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 60–65, San Diego, CA.
- Buades, A., Coll, B., and Morel, J.-M. (2008). Nonlocal image and movie denoising. *International Journal of Computer Vision*, 76(2).
- Buck, I., Finkelstein, A., Jacobs, C., Klein, A., Salesin, D. H., Seims, J., Szeliski, R., and Toyama, K. (2000). Performance-driven hand-drawn animation. In *Symposium on Non Photorealistic Animation and Rendering*, pages 101–108, ACM SIGGRAPH, Annecy.
- Buehler, C., Bosse, M., McMillan, L., Gortler, S. J., and Cohen, M. F. (2001). Unstructured lumigraph rendering. *Proceedings of SIGGRAPH 2001*, , 425–432. ISBN 1-58113-292-1.
- Bugayevskiy, L. M. and Snyder, J. P. (1995). *Map Projections: A Reference Manual*. CRC Press.
- Burl, M. C. and Perona, P. (1996). Recognition of planar object classes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 223–230, San Francisco.
- Burns, P. D. and Williams, D. (1999). Using slanted edge analysis for color registration measurement. In *IS&T PICS Conference*, pages 51–53, Society for Imaging Science and Technology.
- Burt, P. J. (1981). Fast filter transforms for image processing. *Computer Graphics and Image Processing*, 16, 20–51.
- Burt, P. J. and Adelson, E. H. (1983a). The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, COM-31(4), 532–540.
- Burt, P. J. and Adelson, E. H. (1983b). A multiresolution spline with applications to image mosaics. *ACM Transactions on Graphics*, 2(4), 217–236.
- Burt, P. J. and Kolczynski, R. J. (1993). Enhanced image capture through fusion. In *Fourth International Conference on Computer Vision (ICCV'93)*, pages 173–182, IEEE Computer Society Press, Berlin, Germany.
- Can, A. *et al.*. (2002). A feature-based, robust, hierarchical algorithm for registering pairs of images of the curved human retina. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3), 347–364.
- Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), 679–698.
- Capel, D. and Zisserman, A. (1998). Automated mosaicing with super-resolution zoom. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 885–891, Santa Barbara.

- Capel, D. and Zisserman, A. (2000). Super-resolution enhancement of text image sequences. In *Fifteenth International Conference on Pattern Recognition (ICPR'2000)*, pages 600–605, IEEE Computer Society Press, Barcelona, Spain.
- Capel, D. and Zisserman, A. (2001). Super-resolution from multiple views using learnt image models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 627–634, Kauai, Hawaii.
- Caprile, B. and Torre, V. (1990). Using vanishing points for camera calibration. *International Journal of Computer Vision*, 4(2), 127–139.
- Carlstrom, I. et al.. (1992). Modeling and analysis of empirical data in collaborative environments. *Communications of the ACM*, 35(6), 74–84.
- Carneiro, G. and Jepson, A. (2005). The distinctiveness, detectability, and robustness of local image features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 296–301, San Diego, CA.
- Carnevali, P., Coletti, L., and Patarnello, S. (1985). Image processing by simulated annealing. *IBM Journal of Research and Development*, 29(6), 569–579.
- Caselles, V., Kimmel, R., and Sapiro, G. (1997). Geodesic active contours. *International Journal of Computer Vision*, 21(1), 61–79.
- Catmull, E. and Smith, A. R. (1980). 3-d transformations of images in scanline order. *Computer Graphics (SIGGRAPH'80)*, 14(3), 279–285.
- Cham, T. J. and Cipolla, R. (1998). A statistical framework for long-range feature matching in uncalibrated image mosaicing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 442–447, Santa Barbara.
- Cham, T.-J. and Rehg, J. M. (1999). A multiple hypothesis approach to figure tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 239–245, Fort Collins.
- Champleboux, G. et al.. (1992a). Accurate calibration of cameras and range imaging sensors, the NPBS method. In *IEEE International Conference on Robotics and Automation*, pages 1552–1558, IEEE Computer Society Press, Nice, France.
- Champleboux, L., Lavallée, S., Szeliski, R., and Brunie, L. (1992b). From accurate range imaging sensor calibration to accurate model-based 3-D object localization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'92)*, pages 83–89, IEEE Computer Society Press, Champaign, Illinois.
- Chan, T. F. and Vese, L. A. (1992). Active contours without edges. *IEEE Transactions on Image Processing*, 10(2), 266–277.

- Chan, T. F., Osher, S., and Shen, J. (2001). The digital TV filter and nonlinear denoising. *IEEE Transactions on Image Processing*, 10(2), 231–.
- Chaudhuri, S. (2001). *Super-Resolution Imaging*. Springer.
- Cheeseman, P., Kanefsky, B., Hanson, R., and Stutz, J. (1993). *Super-Resolved Surface Reconstruction From Multiple Images*. Technical Report FIA-93-02, NASA Ames Research Center, Artificial Intelligence Branch.
- Chen, C.-Y. and Klette, R. (1999). Image stitching - comparisons and new techniques. In *Computer Analysis of Images and Patterns (CAIP'99)*, pages 615–622, Springer-Verlag, Ljubljana.
- Chen, J., Paris, S., and Durand, F. (2007). Real-time edge-aware image processing with the bilateral grid. *ACM Transactions on Graphics*, 26(3).
- Chen, S. and Williams, L. (1993). View interpolation for image synthesis. *Computer Graphics (SIGGRAPH'93)*, , 279–288.
- Chen, S. E. (1995). QuickTime VR – an image-based approach to virtual environment navigation. *Computer Graphics (SIGGRAPH'95)*, , 29–38.
- Cheng, L., Vishwanathan, S. V. N., and Zhang, X. (2008). Consistent image analogies using semi-supervised learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8), 790–799.
- Chiang, M.-C. and Boult, T. E. (1996). Efficient image warping and super-resolution. In *IEEE Workshop on Applications of Computer Vision (WACV'96)*, pages 56–61, IEEE Computer Society, Sarasota.
- Chou, P. B. and Brown, C. M. (1990). The theory and practice of Bayesian image labeling. *International Journal of Computer Vision*, 4(3), 185–210.
- Christy, S. and Horoud, R. (1996). Euclidean shape and motion from multiple perspective views by affine iterations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11), 1098–1104.
- Chuang, Y.-Y., Curless, B., Salesin, D. H., and Szeliski, R. (2001). A Bayesian approach to digital matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 264–271, Kauai, Hawaii.
- Chuang, Y.-Y., Agarwala, A., Curless, B., Salesin, D. H., and Szeliski, R. (2002). Video matting of complex scenes. *ACM Transactions on Graphics*, 21(3), 243–248.
- Chuang, Y.-Y., Goldman, D. B., Curless, B., Salesin, D. H., and Szeliski, R. (2003). Shadow matting. *ACM Transactions on Graphics*, 22(3), 494–500.

- Chuang, Y.-Y., Goldman, D. B., Zheng, K. C., Curless, B., Salesin, D. H., and Szeliski, R. (2005). Animating pictures with stochastic motion textures. *ACM Transactions on Graphics*, 24(3), 853–860.
- Chuang, Y.-Y., Zongker, D., Hindorff, J., Curless, B., Salesin, D. H., and Szeliski, R. (2000). Environment matting extensions: Towards higher accuracy and real-time capture. In *Computer Graphics (SIGGRAPH'2000 Proceedings)*, pages 121–130, ACM SIGGRAPH, New Orleans.
- Chui, C. K. (1992). *Wavelet Analysis and Its Applications*. Academic Press, New York.
- Chum, O. and Matas, J. (2005). Matching with prosac — progressive sample consensus. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 220–226, San Diego, CA.
- Chum, O. et al.. (2007). Total recall: Automatic query expansion with a generative feature model for object retrieval. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Chum, R. and Matas, J. (2008). *Web Scale Image Clustering*. Technical Report CTU-CMP-2008-15, Center for Machine Perception, Czech Technical University in Prague.
- Cipolla, R. and Blake, A. (1990). The dynamic analysis of apparent contours. In *Third International Conference on Computer Vision (ICCV'90)*, pages 616–623, IEEE Computer Society Press, Osaka, Japan.
- Cipolla, R. and Blake, A. (1992). Surface shape from the deformation of apparent contours. *International Journal of Computer Vision*, 9(2), 83–112.
- Cipolla, R. and Giblin, P. (2000). *Visual Motion of Curves and Surfaces*. Cambridge University Press, Cambridge.
- Cipolla, R. et al.. (1999). Camera calibration from vanishing points in images of architectural scenes. In *British Machine Vision Conference (BMVC)*, Springer-Verlag.
- Clowes, M. B. (1971). On seeing things. *Artificial Intelligence*, 2, 79–116.
- Cochran, S. D. and Medioni, G. G. (1992). 3-d surface description from binocular stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10), 981–994.
- Cohen, L. D. and Cohen, I. (1993). Finite-element methods for active contour models and balloons for 2-D and 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11), 1131–1147.
- Cohen, M. and Wallace, J. (1993). *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann.
- Cohen, M. F. and Szeliski, R. (2006). The Moment Camera. *Computer*, 39(8), 40–45.
- Collins, R. T. (1996). A space-sweep approach to true multi-image matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 358–363, San Francisco.

- Collins, R. T. and Liu, Y. (2003). On-line selection of discriminative tracking features. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 346–352, Nice, France.
- Collins, R. T. and Weiss, R. S. (1990). Vanishing point calculation as a statistical inference on the unit sphere. In *Third International Conference on Computer Vision (ICCV'90)*, pages 400–403, IEEE Computer Society Press, Osaka, Japan.
- Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5), 603–619.
- Comaniciu, D. and Meer, P. (2003). An algorithm for data-driven bandwidth selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2), 281–288.
- Conn, A. R., Gould, N. I. M., and Toint, P. L. (2000). *Trust-Region Methods*. Society for Industrial and Applied Mathematics, Philadelphia.
- Cook, R. L. and Torrance, K. E. (1982). A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1), 7–24.
- Coorg, S. and Teller, S. (2000). Spherical mosaics with quaternions and dense correlation. *International Journal of Computer Vision*, 37(3), 259–273.
- Cootes, T., Edwards, G. J., and Taylor, C. J. (2001). Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6), 681–685.
- Cootes, T., Cooper, D., Taylor, C., and Graham, J. (1995). Active shape models—their training and application. *Computer Vision and Image Understanding*, 61(1), 38–59.
- Cootes, T., Taylor, C., Lanitis, A., Cooper, D., and Graham, J. (1993). Building and using flexible models incorporating grey-level information. In *Fourth International Conference on Computer Vision (ICCV'93)*, pages 242–246, IEEE Computer Society Press, Berlin, Germany.
- Cootes, T. F. and Taylor, C. J. (2001). Statistical models of appearance for medical image analysis and computer vision. In *Medical Imaging*, SPIE.
- Cormen, T. H. (2001). *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts.
- Cornelis, N., Leibe, B., Cornelis, K., and Gool, L. V. (2008). 3d urban scene modeling integrating recognition and reconstruction. *International Journal of Computer Vision*, 78(2-3), 121–141.
- Corso, J. and Hager, G. (2005). Coherent regions for concise and stable image description. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 184–190, San Diego, CA.
- Costeira, J. and Kanade, T. (1995). A multi-body factorization method for motion analysis. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 1071–1076, Cambridge, Massachusetts.

- Cox, I. J. (1994). A maximum likelihood N-camera stereo algorithm. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 733–739, IEEE Computer Society, Seattle.
- Cox, I. J. *et al.*. (1996). A maximum likelihood stereo algorithm. *Computer Vision and Image Understanding*, 63(3), 542–567.
- Cox, I. J., Roy, S., and Hingorani, S. L. (1995). Dynamic histogram warping of image pairs for constant image brightness. In *IEEE International Conference on Image Processing (ICIP'95)*, pages 366–369, IEEE Computer Society.
- Crane, R. (1997). *A Simplified Approach to Image Processing*. Prentice Hall, Upper Saddle River, NJ.
- Cremers, D. *et al.*. (2007). A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape. *International Journal of Computer Vision*, 72(2), 195–215.
- Crevier, D. (1993). *AI: The Tumultuous Search for Artificial Intelligence*. BasicBooks, New York, NY.
- Criminisi, A. *et al.*. (2003). *Efficient Dense-Stereo and Novel-view Synthesis for Gaze Manipulation in One-to-one Teleconferencing*. Technical Report MSR-TR-2003-59, Microsoft Research.
- Criminisi, A. *et al.*. (2006). Bilayer segmentation of live video. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 53–60, New York City, NY.
- Criminisi, A., Pérez, P., and Toyama, K. (2004). Region filling and object removal by exemplar-based inpainting. *IEEE Transactions on Image Processing*, 13(9), 1200–1212.
- Criminisi, A., Reid, I., and Zisserman, A. (2000). Single view metrology. *International Journal of Computer Vision*, 40(2), 123–148.
- Criminisi, A., Shotton, J., Blake, A., and Torr, P. (2003). Gaze manipulation for one-to-one teleconferencing. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 191–198, Nice, France.
- Criminisi, A., Kang, S. B., Swaminathan, R., Szeliski, R., and Anandan, P. (2005). Extracting layers and analyzing their specular properties using epipolar-plane-image analysis. *Computer Vision and Image Understanding*, 97(1), 51–85.
- Crow, F. C. (1984). Summed-area table for texture mapping. *Computer Graphics (SIGGRAPH'84)*, 18(3), 207–212.
- Crowley, J. L. and Stern, R. M. (1984). Fast computation of the difference of low-pass transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(2), 212–222.
- Csurka, G. *et al.*. (2004). Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, Springer-Verlag, Prague.

- Csurka, G. *et al.*. (2007). Generic visual categorization using weak geometry. In Ponce, J. *et al.*, editors, *Toward Category-Level Object Recognition*, pages 207–224, Springer, New York.
- Cui, J., Yang, Q., Wen, F., Wu, Q., Zhang, C., Gool, L. V., and Tang, X. (2008). Transductive object cutout. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Culbertson, B., Malzbender, T., and Slabaugh, G. (1999). Generalized voxel coloring. In *International Workshop on Vision Algorithms*, pages 100–114, Springer, Kerkyra, Greece.
- Curless, B. (1999). From range scans to 3D models. *Computer Graphics*, 33(4), 38–41.
- Curless, B. and Levoy, M. (1995). Better optical triangulation through spacetime analysis. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 987–994, Cambridge, Massachusetts.
- Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Computer Graphics Proceedings, Annual Conference Series*, pages 303–312, ACM SIGGRAPH, Proc. SIGGRAPH'96 (New Orleans).
- Cutler, R. and Turk, M. (1998). View-based interpretation of real-time optical flow for gesture recognition. In *IEEE International Conference on Automatic Face and Gesture Recognition*, pages 416–421, Nara, Japan.
- Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 886–893, San Diego, CA.
- Dana, K. J. *et al.*. (1999). Reflectance and texture of real world surfaces. *ACM Transactions on Graphics*, 18(1), 1–34.
- Danielsson, P. E. (1980). Euclidean distance mapping. *Computer Graphics and Image Processing*, 14(3), 227–248.
- Darrell, T. and Pentland, A. (1991). Robust estimation of a multi-layered motion representation. In *IEEE Workshop on Visual Motion*, pages 173–178, IEEE Computer Society Press, Princeton, New Jersey.
- Davis, J. (1998). Mosaics of scenes with moving objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 354–360, Santa Barbara.
- Davis, L. (1975). A survey of edge detection techniques. *Computer Graphics and Image Processing*, 4(3), 248–270.
- Davison, A. J. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1403–1410, Nice, France.

- De Bonet, J. S. and Viola, P. (1999). Poxels: Probabilistic voxelized volume reconstruction. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 418–425, Kerkyra, Greece.
- de Agapito, L., Hartley, R. I., and Hayman, E. (1999). Linear calibration of a rotating and zooming camera. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 15–21, Fort Collins.
- de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2006). *Computational Geometry: Algorithms and Applications*. Springer, New York, NY, third edition.
- De Bonet, J. (1997). Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics (SIGGRAPH'97)*, , 361–368.
- De Castro, E. and Morandi, C. (1987). Registration of translated and rotated images using finite fourier transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9(5)*, 700–703.
- Debevec, P. (1998). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. *Proceedings of SIGGRAPH 98*, , 189–198. ISBN 0-89791-999-8. Held in Orlando, Florida.
- Debevec, P. (2006). Virtual cinematography: Relighting through computation. *Computer*, 39(8), 57–65.
- Debevec, P., Hawkins, T., Tchou, C., Duiker, H.-P., Sarokin, W., and Sagar, M. (2000). Acquiring the reflectance field of a human face. *Proceedings of SIGGRAPH 2000*, , 145–156.
- Debevec, P. E. (1999). Image-based modeling and lighting. *Computer Graphics*, 33(4), 46–50.
- Debevec, P. E. and Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. *Proceedings of SIGGRAPH 97*, , 369–378. ISBN 0-89791-896-7. Held in Los Angeles, California.
- Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics (SIGGRAPH'96)*, , 11–20.
- Debevec, P. E., Yu, Y., and Borshukov, G. D. (1998). Efficient view-dependent image-based rendering with projective texture-mapping. *Eurographics Rendering Workshop 1998*, , 105–116. ISBN 3-211-83213-0. Held in Vienna, Austria.
- DeCarlo, D. and Santella, A. (2002). Stylization and abstraction of photographs. *ACM Transactions on Graphics*, 21(3), 769–776. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).
- DeCarlo, D., Metaxas, D., and Stone, M. (1998). An anthropometric face model using variational techniques. *Proceedings of SIGGRAPH 98*, , 67–74. ISBN 0-89791-999-8. Held in Orlando, Florida.
- Delingette, H., Hebert, M., and Ikeuchi, K. (1991). Shape representation and image segmentation using deformable surfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 467–472, IEEE Computer Society Press, Maui, Hawaii.

- Dellaert, F. and Collins, R. (1999). Fast image-based tracking by selective pixel integration. In *ICCV Workshop on Frame-Rate Vision*, pages 1–22.
- DeMenthon, D. I. and Davis, L. S. (1995). Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1), 123–141.
- Dempster, A., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39(1), 1–38.
- Deriche, R. (1987). Using canny’s criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, 1(2), 167–187.
- Deriche, R. (1990). Fast algorithms for low-level vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1), 78–87.
- Dev, P. (1974). *Segmentation Processes in Visual Perception: A Cooperative Neural Model*. COINS Technical Report 74C-5, University of Massachusetts at Amherst.
- Dhond, U. R. and Aggarwal, J. K. (1989). Structure from stereo—a review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6), 1489–1510.
- Dick, A., Torr, P., Ruffle, S., and Cipolla, R. (2001). Combining single view recognition and multiple view stereo for architectural scenes. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 268–274, Vancouver, Canada.
- Dickmanns, E. D. and Graefe, V. (1988). Dynamic monocular machine vision. *Machine Vision and Applications*, 1, 223–240.
- Diebel, J. (2006). *Representing Attitude: Euler Angles, Quaternions, and Rotation Vectors*. Technical Report, Stanford University. <http://ai.stanford.edu/~diebel/attitude.html>.
- Dodgson, N. A. (1992). *Image Resampling*. Technical Report TR261, Wolfson College and Computer Laboratory, University of Cambridge.
- Duda, R. O. and Hart, P. E. (1972). Use of the hough transform to detect lines and curves in pictures. *Communications of the ACM*, 15(1), 11–15.
- Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. John Wiley & Sons, New York, 2nd edition.
- Duff, I. S., editor. (1981). *IMA Numerical Analysis Group Conference*, Academic Press.
- Dupuis, P. and Oliensis, J. (1994). An optimal control formulation and related numerical methods for a problem in shape reconstruction. *Annals of Applied Probability*, 4(2), 287–346.

- Durand, F. and Dorsey, J. (2002). Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics (TOG)*, 21(3), 257–266.
- Durand, F. and Szeliski, R. (2007). Computational photography. *IEEE Computer Graphics and Applications*, 27(2), 21–22. Guest Editors’ Introduction to Special Issue.
- Durbin, R. and Willshaw, D. (1987). An analogue approach to the traveling salesman problem using an elastic net method. *Nature*, 326, 689–691.
- Durbin, R., Szeliski, R., and Yuille, A. (1989). An analysis of the elastic net approach to the travelling salesman problem. *Neural Computation*, 1(3), 348–358.
- Eden, A., Uyttendaele, M., and Szeliski, R. (2006). Seamless image stitching of scenes with large motions and exposure differences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’2006)*, pages 2498–2505, New York, NY.
- Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 341–346, ACM Press / ACM SIGGRAPH.
- Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *Seventh International Conference on Computer Vision (ICCV’99)*, pages 1033–1038, Kerkyra, Greece.
- Eisemann, E. and Durand, F. (2004). Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics*, 23(3), 673–?
- Eisert, P., Fechteler, P., and Rurainsky, J. (2008). 3d tracking of shoes for virtual mirror applications. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Eisert, P., Steinbach, E., and Girod, B. (2000a). Automatic reconstruction of stationary 3-D objects from multiple uncalibrated camera views. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(2), 261–277.
- Eisert, P., Wiegand, T., and Girod, B. (2000b). Model-aided coding: a new approach to incorporate facial animation into motion-compensated video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(3), 344–358.
- El-Melegy, M. and Farag, A. (2003). Nonmetric lens distortion calibration: Closed-form solutions, robust estimation and model selection. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 554–559, Nice, France.
- Elder, J. H. (1999). Are edges incomplete? *International Journal of Computer Vision*, 34(2/3), 97–122.

- Elder, J. H. and Goldberg, R. M. (2001). Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3), 291–296.
- Elder, J. H. and Zucker, S. W. (1998). Local scale control for edge detection and blur estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(7), 699–716.
- Fairchild, M. (1998). *Color Appearance Models*. Addison-Wesley, Reading, MA.
- Fairchild, M. D. (2005). *Color Appearance Models*. Wiley, 2nd edition.
- Farbman, Z., Fattal, R., Lischinski, D., and Szeliski, R. (2008). Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics*, 27(3).
- Farin, G. (1992). From conics to NURBS: A tutorial and survey. *IEEE Computer Graphics and Applications*, 12(5), 78–86.
- Farin, G. E. (1996). *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, Boston, Massachusetts, 4th edition.
- Fattal, R., Lischinski, D., and Werman, M. (2002). Gradient domain high dynamic range compression. *ACM Transactions on Graphics (TOG)*, 21(3), 249–256.
- Faugeras, O. (1993). *Three-dimensional computer vision: A geometric viewpoint*. MIT Press, Cambridge, Massachusetts.
- Faugeras, O. and Keriven, R. (1998). Variational principles, surface evolution, PDEs, level set methods, and the stereo problem. *IEEE Transactions on Image Processing*, 7(3), 335–344.
- Faugeras, O. and Luong, Q.-T. (2001). *The Geometry of Multiple Images*. MIT Press, Cambridge, MA.
- Faugeras, O. D. (1992). What can be seen in three dimensions with an uncalibrated stereo rig? In *Second European Conference on Computer Vision (ECCV'92)*, pages 563–578, Springer-Verlag, Santa Margherita Liguere, Italy.
- Faugeras, O. D. and Hebert, M. (1987). The representation, recognition and positioning of 3-D shapes from range data. In Kanade, T., editor, *Three-Dimensional Machine Vision*, pages 301–353, Kluwer Academic Publishers, Boston.
- Favaro, P., Duci, A., Ma, Y., and Soatto, S. (2003). On exploiting occlusions in multiple-view geometry. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 479–486, Nice, France.
- Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27, 861–874.
- Fei-Fei, L. (2007). Bag-of-word models. In *Recognizing and Learning Object Categories*, page CVPR 2007 Short Course on Recognizing and Learning Object Categories. <http://people.csail.mit.edu/torralba/shortCourseRLOC/>.

- Fei-Fei, L., Fergus, R., and Perona, P. (2006). One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4), 594–611.
- Fei-Fei, L., Fergus, R., and Torralba, A. (2007). Recognizing and learning object categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, page Short Course, Minneapolis, MN. <http://people.csail.mit.edu/torralba/shortCourseRLOC/>.
- Feilner, M. et al.. (2005). An orthogonal family of quincunx wavelets with continuously adjustable order. *IEEE Transactions on Image Processing*, 14(4), 499–520.
- Felzenszwalb, P., McAllester, D., and Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2004a). Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), 167–181.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2004b). Efficient belief propagation for early vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2004)*, pages 261–268, IEEE Computer Society, Washington, DC.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1), 55–79.
- Ferencz, A. et al.. (2008). Learning to locate informative features for visual identification. *International Journal of Computer Vision*, 77(1-3), 3–24.
- Fergus, R. (2007a). Combined segmentation and recognition. In *Recognizing and Learning Object Categories*, page CVPR 2007 Short Course on Recognizing and Learning Object Categories. <http://people.csail.mit.edu/torralba/shortCourseRLOC/>.
- Fergus, R. (2007b). Part-based models. In *Recognizing and Learning Object Categories*, page CVPR 2007 Short Course on Recognizing and Learning Object Categories. <http://people.csail.mit.edu/torralba/shortCourseRLOC/>.
- Fergus, R., Perona, P., and Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2003)*, pages 264–271, Madison, WI.
- Fergus, R., Perona, P., and Zisserman, A. (2007). Weakly supervised scale-invariant learning of models for visual recognition. *International Journal of Computer Vision*, 71(3), 273–303.

- Fergus, R., Fei-Fei, L., Perona, P., and Zisserman, A. (2005). Learning object categories from Google's image search. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1816–1823, Beijing, China.
- Fergus, R., Singh, B., Hertzmann, A., Roweis, S. T., and Freeman, W. T. (2006). Removing camera shake from a single photograph. *ACM Transactions on Graphics*, 25(3), 787–794.
- Finkelstein, A. and Salesin, D. H. (1994). Multiresolution curves. In *Proceedings of SIGGRAPH 94*, pages 261–268.
- Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 381–395.
- Fischler, M. A. and Elschlager, R. A. (1973). The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 22(1), 67–92.
- Fischler, M. A. and Firschein, O. (1987). *Readings in Computer Vision*. Morgan Kaufmann Publishers, Inc., Los Altos.
- Fischler, M. A., Firschein, O., Barnard, S. T., Fua, P. V., and Leclerc, Y. (1989). *The Vision Problem: Exploiting Parallel Computation*. Technical Note 458, SRI International, Menlo Park.
- Fitzgibbon, A., Wexler, Y., and Zisserman, A. (2003). Image-based rendering using image-based priors. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1176–1183, Nice, France.
- Fitzgibbon, A. W. and Zisserman, A. (1998). Automatic camera recovery for closed and open image sequences. In *Fifth European Conference on Computer Vision (ECCV'98)*, pages 311–326, Springer-Verlag, Freiburg, Germany.
- Fitzgibbon, A. W., Cross, G., and Zisserman, A. (1998). Automatic 3D model construction for turn-table sequences. In Koch, R. and Van Gool, L., editors, *European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE)*, pages 155–170, Springer-Verlag, Freiburg.
- Fleet, D. and Jepson, A. (1990). Computation of component image velocity from local phase information. *International Journal of Computer Vision*, 5(1), 77–104.
- Fleet, D. J. (1992). *Measurement of Image Velocity*. Kluwer Academic Publishers, Boston.
- Fleet, D. J., Jepson, A. D., and Jenkin, M. R. M. (1991). Phase-based disparity measurement. *CVGIP: Image Understanding*, 53(2), 198–210.
- Fletcher, R. (1980). *Conjugate Direction Methods in Optimization*. John Wiley & Sons, Chichester, England.
- Flickner, M. et al.. (1995). Query by image and video content: The qbic system. *Computer*, 28(9), 23–32.

- Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1995). *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 2 edition.
- Förstner, W. (1986). A feature-based correspondence algorithm for image matching. *Intl. Arch. Photogrammetry & Remote Sensing*, 26(3), 150–166.
- Förstner, W. (1994). A framework for low level feature extraction. In *Third European Conference on Computer Vision (ECCV'94)*, pages 383–394, Springer-Verlag, Stockholm, Sweden.
- Förstner, W. (2005). Uncertainty and projective geometry. In Bayro-Corrochano, E., editor, *Handbook of Geometric Computing*, pages 493–534, Springer, New York.
- Forsyth, D. *et al.*. (2005). Computational studies of human motion. *Foundations and Trends in Computer Graphics and Computer Vision*, 1(2).
- Forsyth, D. and Ponce, J. (2003). *Computer Vision: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.
- Forsyth, D. A. and Fleck, M. M. (1999). Automatic detection of human nudes. *International Journal of Computer Vision*, 32(1), 63–77.
- Fowlkes, C. C. *et al.*. (2004). Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2), 214–225.
- Frahm, J.-M. and Koch, R. (2003). Camera calibration with known rotation. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1418–1425, Nice, France.
- Freeman, M. (2008). *Mastering HDR Photography*. Amphoto Books, New York.
- Freeman, W. T. (1992). *Steerable Filters and Local Analysis of Image Structure*. Ph.D. thesis, Massachusetts Institute of Technology.
- Freeman, W. T., Anderson, D. B., Beardsley, P. A., Dodge, C. N., Roth, M., Weissman, C. D., Yerazunis, W. S., Kage, H., Kyuma, K., Miyake, Y., and Tanaka, K. (1998). Computer vision for interactive computer graphics. *IEEE Computer Graphics and Applications*, 18(3), 42–53.
- Freeman, W. T. and Adelson, E. H. (1991). The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9), 891–906.
- Freeman, W. T., Jones, T. R., and Pasztor, E. C. (2002). Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2), 56–65.
- Freeman, W. T., Pasztor, E. C., and Carmichael, O. T. (2000). Learning low-level vision. *International Journal of Computer Vision*, 40(1), 25–47.

- Freeman, W. T., Beardsley, P. A., Kage, H., Tanaka, K.-I., Kyuma, K., and Weissman, C. D. (1999). Computer vision for computer interaction. *Computer Graphics*, 33(4), 65–68.
- Friskin, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R. (2000). Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of SIGGRAPH 2000*, , 249–254. ISBN 1-58113-208-5.
- Fritz, M. and Schiele, B. (2008). Decomposition, discovery and detection of visual categories using topic models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Fua, P. (1993). A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6(1), 35–49.
- Fua, P. and Leclerc, Y. G. (1995). Object-centered surface reconstruction: Combining multi-image stereo and shading. *International Journal of Computer Vision*, 16, 35–56.
- Fua, P. and Sander, P. (1992). Segmenting unstructured 3D points into surfaces. In *Second European Conference on Computer Vision (ECCV'92)*, pages 676–680, Springer-Verlag, Santa Margherita Liguere, Italy.
- Fuh, C.-S. and Maragos, P. (1991). Motion displacement estimation using an affine model for image matching. *Optical Engineering*, 30(7), 881–887.
- Fukunaga, K. and Hostetler, L. D. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21, 32–40.
- Furukawa, Y. and Ponce, J. (2007). Accurate, dense, and robust multi-view stereopsis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Furukawa, Y. and Ponce, J. (2008a). Accurate calibration from multi-view stereo and bundle adjustment. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Furukawa, Y. and Ponce, J. (2008b). Dense 3d motion capture from synchronized video streams. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Gallagher, A. C. and Chen, T. (2008). Multi-image graph cut clothing segmentation for recognizing people. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Gallup, D., Frahm, J.-M., Mordohai, P., and Pollefeys, M. (2008). Variable baseline/resolution stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.

- Gamble, E. and Poggio, T. (1987). *Visual integration and detection of discontinuities: the key role of intensity edges*. A. I. Memo 970, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.
- Geiger, D. and Girosi, F. (1991). Parallel and deterministic algorithms for MRF's: Surface reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5), 401–412.
- Geiger, D., Ladendorf, B., and Yuille, A. (1992). Occlusions and binocular stereo. In *Second European Conference on Computer Vision (ECCV'92)*, pages 425–433, Springer-Verlag, Santa Margherita Liguere, Italy.
- Gelb, A., editor. (1974). *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts.
- Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6), 721–741.
- Gennert, M. A. (1988). Brightness-based stereo matching. In *Second International Conference on Computer Vision (ICCV'88)*, pages 139–143, IEEE Computer Society Press, Tampa.
- Gill, P. E., Murray, W., and Wright, M. H. (1981). *Practical Optimization*. Academic Press, Inc., London, England.
- Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *25th International Conference on Very Large Data Bases (VLDB'99)*, pages 518–529.
- Girod, B., Greiner, G., and Niemann, H., editors. (2000). *Principles of 3D Image Analysis and Synthesis*, Kluwer, Boston.
- Glassner, A. S. (1995). *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers, San Francisco.
- Gleicher, M. (1995). Image snapping. *Proceedings of SIGGRAPH 95*, , 183–190. ISBN 0-201-84776-0. Held in Los Angeles, California.
- Gleicher, M. (1999). Animation from observation: Motion capture and motion editing. *Computer Graphics (SIGGRAPH'99)*, 33(4), 51–54.
- Gleicher, M. and Witkin, A. (1992). Through-the-lens camera control. *Computer Graphics (SIGGRAPH'92)*, 26(2), 331–340.
- Glocker, B., Paragios, N., Komodakis, N., Tziritas, G., and Navab, N. (2008). Optical flow estimation with uncertainties through dynamic mrf's. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Gluckman, J. (2006a). Higher order image pyramids. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 308–320, Springer.

- Gluckman, J. (2006b). Scale variant image pyramids. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 1069–1075, New York City, NY.
- Goesele, M. et al.. (2003). Accuracy of 3d range scanners by measurement of the slanted edge modulation transfer function. In *Fourth International Conference on 3-D Digital Imaging and Modeling*, Banff.
- Goesele, M. et al.. (2007). Multi-view stereo for community photo collections. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *Journal of the Association for Computing Machinery*, 35(4), 921–940.
- Goldman, D. B. and Chen, J.-H. (2005). Vignette and exposure calibration and compensation. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 899–906, Beijing, China.
- Golub, G. and Van Loan, C. F. (1996). *Matrix Computation, third edition*. The John Hopkins University Press, Baltimore and London.
- Gomes, J. and Velho, L. (1997). *Image Processing for Computer Graphics*. Springer-Verlag, New York.
- Gomes, J., Darsa, L., Costa, B., and Velho, L. (1999). *Warping and Morphing of Graphical Objects*. Morgan Kaufmann Publishers, San Francisco.
- Gonzales, R. C. and Woods, R. E. (2002). *Digital Image Processing*. Addison-Wesley.
- Gooch, B. and Gooch, A. (2001). *Non-Photorealistic Rendering*. A K Peters, Ltd, Natick, Massachusetts.
- Gortler, S. J. and Cohen, M. F. (1995). Hierarchical and variational geometric modeling with wavelets. In *Symposium on Interactive 3D Graphics*, pages 35–43, Monterey, CA.
- Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The Lumigraph. In *Computer Graphics Proceedings, Annual Conference Series*, pages 43–54, ACM SIGGRAPH, Proc. SIGGRAPH'96 (New Orleans).
- Goshtasby, A. (1989). Correction of image deformation from lens distortion using bezier patches. *Computer Vision, Graphics, and Image Processing*, 47(4), 385–394.
- Goshtasby, A. (2005). *2-D and 3-D Image Registration*. Wiley, New York.
- Govindu, V. M. (2006). Revisiting the brightness constraint: Probabilistic formulation and algorithms. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 177–188, Springer.
- Grady, L. (2006). Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11), 1768–1783.

- Grady, L. (2008). A lattice-preserving multigrid method for solving the inhomogeneous poisson equations used in image analysis. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 252–264, Springer-Verlag, Marseilles.
- Grady, L. and Ali, S. (2008). Fast approximate random walker segmentation using eigenvector precomputation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Grady, L. and Alvino, C. (2008). Reformulating and optimizing the Mumford-Shah functional on a graph — a faster, lower energy solution. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 248–261, Springer-Verlag, Marseilles.
- Grauman, K. and Darrell, T. (2005). Efficient image matching with distributions of local invariant features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 627–634, San Diego, CA.
- Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11), 21–29.
- Greene, N. and Heckbert, P. (1986). Creating raster Omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications*, 6(6), 21–27.
- Greig, D., Porteous, B., and Seheult, A. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2), 271–279.
- Greban, K. D., Thorpe, C. E., and Kanade, T. (1988). Geometric camera calibration using systems of linear equations. In *IEEE International Conference on Robotics and Automation*, pages 562–567, IEEE Computer Society Press, Philadelphia.
- Grimson, W. E. L. (1985). Computational experiments with a feature based stereo algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7(1)*, 17–34.
- Gross, R., Sweeney, L., la Torre, F. D., and Baker, S. (2008). Semi-supervised learning of multi-factor models for face de-identification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Grossberg, M. D. and Nayar, S. K. (2001). A general imaging model and a method for finding its parameters. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 108–115, Vancouver, Canada.
- Grossberg, M. D. and Nayar, S. K. (2004). Modeling the space of camera response functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10), 1272–1282.
- Guenter, B. *et al.*. (1998). Making faces. *Proceedings of SIGGRAPH 98*, , 55–66.

- Habbecke, M. and Kobbelt, L. (2007). A surface-growing approach to multi-view stereo reconstruction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Hager, G. D. and Belhumeur, P. N. (1998). Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10), 1025–1039.
- Hall, R. (1989). *Illumination and Color in Computer Generated Imagery*. Springer-Verlag, New York.
- Hampel, F. R. et al.. (1986). *Robust Statistics : The Approach Based on Influence Functions*. Wiley, New York.
- Han, F. and Zhu, S.-C. (2005). Bottom-up/top-down image parsing by attribute graph grammar. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1778–1785, Beijing, China.
- Hannah, M. J. (1974). *Computer Matching of Areas in Stereo Images*. Ph.D. thesis, Stanford University.
- Hannah, M. J. (1988). Test results from SRI's stereo system. In *Image Understanding Workshop*, pages 740–744, Morgan Kaufmann Publishers, Cambridge, Massachusetts.
- Hansen, M., Anandan, P., Dana, K., van der Wal, G., and Burt, P. (1994). Real-time scene stabilization and mosaic construction. In *IEEE Workshop on Applications of Computer Vision (WACV'94)*, pages 54–62, IEEE Computer Society, Sarasota.
- Hanson, A. R. and Riseman, E. M., editors. (1978). *Computer Vision Systems*, Academic Press, New York.
- Haralick, R. M. and Shapiro, L. G. (1985). Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1), 100–132.
- Haralick, R. M. and Shapiro, L. G. (1992). *Computer and Robot Vision*. Addison-Wesley, Reading, MA.
- Harker, M. and O'Leary, P. (2008). Least squares surface reconstruction from measured gradient fields. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Harris, C. and Stephens, M. J. (1988). A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–152.
- Hartley, R., Gupta, R., and Chang, T. (1992). Estimation of relative camera positions for uncalibrated cameras. In *Second European Conference on Computer Vision (ECCV'92)*, pages 579–587, Springer-Verlag, Santa Margherita Liguere, Italy.
- Hartley, R. I. (1994a). An algorithm for self calibration from several views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 908–912, IEEE Computer Society, Seattle.

- Hartley, R. I. (1994b). Projective reconstruction and invariants from multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10), 1036–1041.
- Hartley, R. I. (1994c). Self-calibration from multiple views of a rotating camera. In *Third European Conference on Computer Vision (ECCV'94)*, pages 471–478, Springer-Verlag, Stockholm, Sweden.
- Hartley, R. I. (1997). Self-calibration of stationary cameras. *International Journal of Computer Vision*, 22(1), 5–23.
- Hartley, R. I. (1998). Chirality. *International Journal of Computer Vision*, 26(1), 41–61.
- Hartley, R. I. and Sturm, P. (1997). Triangulation. *Computer Vision and Image Understanding*, 68(2), 146–157.
- Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry*. Cambridge University Press, Cambridge, UK.
- Hartley, R. I., Hayman, E., de Agapito, L., and Reid, I. (2000). Camera calibration and the search for infinity. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 510–517, Hilton Head Island.
- Hasinoff, S. W., Kang, S. B., and Szeliski, R. (2006). Boundary matting for view synthesis. *Computer Vision and Image Understanding*, 103(1), 22–32.
- Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York.
- Hayes, B. (2008). Computational photography. *American Scientist*, 96, 94–99.
- Hays, J. and Efros, A. A. (2007). Scene completion using millions of photographs. *ACM Transactions on Graphics*, 26(3).
- He, L.-W. and Zhang, Z. (2005). Real-time whiteboard capture and processing using a video camera for teleconferencing. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*, pages 1113–1116, Philadelphia.
- Healey, G. E. and Kondepudy, R. (1994). Radiometric CCD camera calibration and noise estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3), 267–276.
- Healey, G. E. and Shafer, S. A. (1992). *Color. Physics-Based Vision: Principles and Practice*, Jones & Bartlett, Cambridge, MA.
- Heath, M. D., Sarkar, S., Sanocki, T., and Bowyer, K. W. (1998). Comparison of edge detectors. *Computer Vision and Image Understanding*, 69(1), 38–54.
- Hecht, E. (2001). *Optics*. Pearson Addison Wesley, Reading, MA, 4th edition.

- Heckbert, P. (1986). Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11), 56–67.
- Heckbert, P. (1989). *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, The University of California at Berkeley.
- Heeger, D. J. (1988). Optical flow using spatiotemporal filters. *International Journal of Computer Vision*, 1(1), 279–302.
- Heo, Y. S., Lee, K. M., and Lee, S. U. (2008). Illumination and camera invariant stereo matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Herley, C. (2005). Automatic occlusion removal from minimum number of images. In *International Conference on Image Processing (ICIP 2005)*, pages 1046–1049–16, Genova.
- Hernandez, C., Vogiatzis, G., , and Cipolla, R. (2007). Probabilistic visibility for multi-view stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Hertzmann, A. *et al.*. (2001). Image analogies. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 327–340, ACM Press / ACM SIGGRAPH.
- Hillman, P., Hannah, J., and Renshaw, D. (2001). Alpha channel estimation in high resolution images and image sequences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 1063–1068, Kauai, Hawaii.
- Hilton, A. *et al.*. (1996). Reliable surface reconstruction from multiple range images. In *Fourth European Conference on Computer Vision (ECCV'96)*, pages 117–126, Springer-Verlag, Cambridge, England.
- Hinckley, K., Sinclair, M., Hanson, E., Szeliski, R., and Conway, M. (1999). The VideoMouse: a camera-based multi-degree-of-freedom input device. In *12th annual ACM symposium on User interface software and technology*, pages 103–112, ACM SIGGRAPH.
- Hinterstoisser, S., Benhimane, S., Navab, N., Fua, P., and Lepetit, V. (2008). Online learning of patch perspective rectification for efficient object detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Hinton, G. E. (1977). *Relaxation and its Role in Vision*. Ph.D. thesis, University of Edinburgh.
- Hirschmuller, H. and Scharstein, D. (2007). Evaluation of cost functions for stereo matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.

- Hjaltason, G. R. and Samet, H. (2003). Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4), 517–580.
- Hoiem, D., Efros, A. A., and Hebert, M. (2005). Geometric context from a single image. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 654–661, Beijing, China.
- Hoiem, D., Rother, C., , and Winn, J. (2007). 3d layoutcrf for multi-view object class recognition and segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Hoppe, W. *et al.*. (1992). Surface reconstruction from unorganized points. *Computer Graphics (SIGGRAPH'92)*, 26(2), 71–78.
- Hoppe, W. *et al.*. (1993). Mesh optimization. *Computer Graphics (SIGGRAPH'93)*, , 19–25.
- Horn, B. K. P. (1974). Determining lightness from an image. *Computer Graphics and Image Processing*, 3(1), 277–299.
- Horn, B. K. P. (1975). Obtaining shape from shading information. In Winston, P. H., editor, *The Psychology of Computer Vision*, pages 115–155, McGraw-Hill, New York.
- Horn, B. K. P. (1986). *Robot Vision*. MIT Press, Cambridge, Massachusetts.
- Horn, B. K. P. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4), 629–642.
- Horn, B. K. P. (1990). Height and gradient from shading. *International Journal of Computer Vision*, 5(1), 37–75.
- Horn, B. K. P. and Weldon Jr., E. J. (1988). Direct methods for recovering motion. *International Journal of Computer Vision*, 2(1), 51–76.
- Horn, B. K. P. and Brooks, M. J. (1986). The variational approach to shape from shading. *Computer Vision, Graphics, and Image Processing*, 33, 174–208.
- Horn, B. K. P. and Brooks, M. J. (1989). *Shape from Shading*. MIT Press, Cambridge, Massachusetts.
- Horn, B. K. P. and Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, 17, 185–203.
- Hornung, A., Zeng, B., and Kobbelt, L. (2008). Image selection for improved multi-view stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Horry, Y., ichi Anjyo, K., and Arai, K. (1997). Tour into the picture: Using a spidery mesh interface to make animation from a single image. *Proceedings of SIGGRAPH 97*, , 225–232. ISBN 0-89791-896-7. Held in Los Angeles, California.

- Hough, P. V. C. (1962). Method and means for recognizing complex patterns. *U. S. Patent, 3,069,654*.
- Houhou, N., Thiran, J.-P., and Bresson, X. (2008). Fast texture segmentation using the shape operator and active contour. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Hsieh, Y. C., McKeown, D., and Perlant, F. P. (1992). Performance evaluation of scene registration and stereo matching for cartographic feature extraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 14*(2), 214–238.
- Hsu, P. R. and Sawhney, H. S. (1998). Influence of global constraints and lens distortion on pose and appearance recovery from a purely rotating camera. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 154–159, IEEE Computer Society, Princeton.
- Hua, G., Brown, M., and Winder, S. (2007). Discriminant embedding for local image descriptors. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Huang, T. S. (1981). *Image Sequence Analysis*. Springer-Verlag, Berlin, Heidelberg.
- Huber, P. J. (1981). *Robust Statistics*. John Wiley & Sons, New York.
- Huffman, D. A. (1971). Impossible objects and nonsense sentences. *Machine Intelligence, 8*, 295–323.
- Huttenlocher, D. P., Klanderman, G., and Rucklidge, W. (1993). Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 15*(9), 850–863.
- Huynh, D. Q., Hartley, R., and Heyden, A. (2003). Outlier correcton in image sequences for the affine camera. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 585–590, Nice, France.
- Igarashi, T., Nishino, K., and Nayar, S. (2007). The appearance of human skin: A survey. *Foundations and Trends in Computer Graphics and Computer Vision, 3*(1).
- Illingworth, J. and Kittler, J. (1988). A survey of the Hough transform. *Computer Vision, Graphics, and Image Processing, 44*, 87–116.
- Intille, S. S. and Bobick, A. F. (1994). Disparity-space images and large occlusion stereo. In *Proc. Third European Conference on Computer Vision (ECCV'94)*, Springer-Verlag, Stockholm, Sweden.
- Irani, M. and Anandan, P. (1998). Video indexing based on mosaic representations. *Proceedings of the IEEE, 86*(5), 905–921.
- Irani, M. and Anandan, P. (1999). About direct methods. In *International Workshop on Vision Algorithms*, pages 267–277, Springer, Kerkyra, Greece.
- Irani, M. and Peleg, S. (1991). Improving resolution by image registration. *Graphical Models and Image Processing, 53*(3), 231–239.

- Irani, M., Hsu, S., and Anandan, P. (1995). Video compression using mosaic representations. *Signal Processing: Image Communication*, 7, 529–552.
- Irani, M., Rousso, B., and Peleg, S. (1997). Recovery of ego-motion using image stabilization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3), 268–272.
- Isard, M. and Blake, A. (1998). CONDENSATION—conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1), 5–28.
- Ishiguro, H., Yamamoto, M., and Tsuji, S. (1992). Omni-directional stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 257–262.
- Ishikawa, H. (2003). Exact optimization for Markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10), 1333–1336.
- Ishikawa, H. and Geiger, D. (1998). Occlusions, discontinuities, and epipolar lines in stereo. In *Fifth European Conference on Computer Vision (ECCV'98)*, pages 232–248, Springer-Verlag, Freiburg, Germany.
- Isidoro, J. and Sclaroff, S. (2003). Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1335–1342, Nice, France.
- J. S. Wiejak, H. B. and Buxton, B. F. (1985). Convolution with separable masks for early image processing. *Computer Vision, Graphics, and Image Processing*, 32(3), 279–290.
- Jacobs, C. E., Finkelstein, A., and Salesin, D. H. (1995). Fast multiresolution image querying. *Proceedings of SIGGRAPH 95*, , 277–286.
- Jähne, B. (1997). *Digital Image Processing*. Springer-Verlag, Berlin.
- Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey.
- Jain, A. K., Duin, R. P. W., and Mao, J. (2000). Statistical pattern recognition: A review. In *International Conference on Pattern Recognition (ICPR 2004)*, pages 4–37.
- Jain, A. K., Topchy, A., Law, M. H. C., and Buhmann, J. M. (2004). Landscape of clustering algorithms. In *International Conference on Pattern Recognition (ICPR 2004)*, pages 260–263.
- Jenkin, M. R. M., Jepson, A. D., and Tsotsos, J. K. (1991). Techniques for disparity measurement. *CVGIP: Image Understanding*, 53(1), 14–30.
- Jia, J. and Tang, C.-K. (2003). Image registration with global and local luminance alignment. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 156–163, Nice, France.

- Jia, J., Sun, J., Tang, C.-K., and Shum, H.-Y. (2006). Drag-and-drop pasting. *ACM Transactions on Graphics*, 25(3), 631–636.
- Johnson, A. E. and Kang, S. B. (1997). Registration and integration of textured 3-D data. In *International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, Ottawa.
- Jojic, N. and Frey, B. J. (2001). Learning flexible sprites in video layers. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 199–206, Kauai, Hawaii.
- Jones, D. G. and Malik, J. (1992). A computational framework for determining stereo correspondence from a set of linear spatial filters. In *Second European Conference on Computer Vision (ECCV'92)*, pages 397–410, Springer-Verlag, Santa Margherita Liguere, Italy.
- Jones, M. J. and Rehg, J. M. (2001). Automatic detection of human nudes. *International Journal of Computer Vision*, 46(1), 81–96.
- Jordan, M. I. *et al.*. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2), 183–233.
- Joshi, N., Matusik, W., and Avidan, S. (2006). Natural video matting using camera arrays. *ACM Transactions on Graphics*, 25(3), 779–786.
- Joshi, N., Szeliski, R., and Kriegman, D. J. (2008). PSF estimation using sharp edge prediction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Ju, S. X., Black, M. J., and Jepson, A. D. (1996). Skin and bones: Multi-layer, locally affine, optical flow and regularization with transparency. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 307–314, San Francisco.
- Jurie, F. and Dhome, M. (2002). Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7), 996–1000.
- Kadir, T., Zisserman, A., and Brady, M. (2004). An affine invariant salient region detector. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 228–241, Springer-Verlag, Prague.
- Kaftory, R., Schechner, Y., and Zeevi, Y. (2007). Variational distance-dependent image restoration. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Kakumanu, P. *et al.*. (2007). A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40(3), 1106–1122.
- Kamvar, S. D., Klein, D., and Manning, C. D. (2002). Interpreting and extending classical agglomerative clustering algorithms using a model-based approach. In *International Conference on Machine Learning*, pages 283–290.

- Kanade, T. (1977). *Computer Recognition of Human Faces*. Birkhauser, Basel.
- Kanade, T. (1980). A theory of the origami world. *Artificial Intelligence*, 13, 279–311.
- Kanade, T., editor. (1987). *Three-Dimensional Machine Vision*, Kluwer Academic Publishers, Boston.
- Kanade, T. (1994). Development of a video-rate stereo machine. In *Image Understanding Workshop*, pages 549–557, Morgan Kaufmann Publishers, Monterey.
- Kanade, T. and Okutomi, M. (1994). A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9), 920–932.
- Kanade, T. et al.. (1996). A stereo machine for video-rate dense depth mapping and its new applications. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 196–202, San Francisco.
- Kanade, T., Rander, P. W., and Narayanan, P. J. (1997). Virtualized reality: constructing virtual worlds from real scenes. *IEEE MultiMedia Magazine*, 1(1), 34–47.
- Kang, S. B. (1999). A survey of image-based rendering techniques. In *Videometrics VI*, pages 2–16, SPIE, San Jose.
- Kang, S. B. (2001). Radial distortion snakes. *IEICE Trans. Inf. & Syst.*, E84-D(12), 1603–1611.
- Kang, S. B. and Szeliski, R. (1997). 3-D scene data recovery using omnidirectional multibaseline stereo. *International Journal of Computer Vision*, 25(2), 167–183.
- Kang, S. B. and Szeliski, R. (2004). Extracting view-dependent depth maps from a collection of images. *International Journal of Computer Vision*, 58(2), 139–163.
- Kang, S. B. and Weiss, R. (1997). Characterization of errors in compositing panoramic images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 103–109, San Juan, Puerto Rico.
- Kang, S. B. and Weiss, R. (1999). Characterization of errors in compositing panoramic images. *Computer Vision and Image Understanding*, 73(2), 269–280.
- Kang, S. B. and Weiss, R. (2000). Can we calibrate a camera using an image of a flat, textureless Lambertian surface? In *Sixth European Conference on Computer Vision (ECCV 2000)*, pages 640–653, Dublin, Ireland.
- Kang, S. B., Szeliski, R., and Anandan, P. (2000). The geometry-image representation tradeoff for rendering. In *International Conference on Image Processing (ICIP-2000)*, pages 13–16, Vancouver.
- Kang, S. B., Szeliski, R., and Chai, J. (2001). Handling occlusions in dense multi-view stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 103–110, Kauai, Hawaii.

- Kang, S. B., Szeliski, R., and Shum, H.-Y. (1997). A parallel feature tracker for extended image sequences. *Computer Vision and Image Understanding*, 67(3), 296–310.
- Kang, S. B., Szeliski, R., and Uyttendaele, M. (2004). *Seamless Stitching using Multi-Perspective Plane Sweep*. Technical Report MSR-TR-2004-48, Microsoft Research.
- Kang, S. B., Li, Y., Tong, X., and Shum, H.-Y. (2006). Image-based rendering. *Foundations and Trends in Computer Graphics and Computer Vision*, 2(3).
- Kang, S. B., Uyttendaele, M., Winder, S., and Szeliski, R. (2003). High dynamic range video. *ACM Transactions on Graphics*, 22(3), 319–325.
- Kang, S. B., Webb, J., Zitnick, L., and Kanade, T. (1995). A multibaseline stereo system with active illumination and real-time image acquisition. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 88–93, Cambridge, Massachusetts.
- Kannala, J., Rahtu, E., Brandt, S. S., and Heikkila, J. (2008). Object recognition and segmentation by non-rigid quasi-dense matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Kass, M. (1988). Linear image features in stereopsis. *International Journal of Computer Vision*, 1(4), 357–368.
- Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1(4), 321–331.
- Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Hoboken.
- Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, pages 61–70.
- Ke, Y. and Sukthankar, R. (2004). PCA-SIFT: a more distinctive representation for local image descriptors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2004)*, pages 506–513, Washington, DC.
- Kenney, C., Zuliani, M., and Manjunath, B. (2005). An axiomatic approach to corner detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 191–197, San Diego, CA.
- Keren, D., Peleg, S., and Brada, R. (1988). Image sequence enhancement using sub-pixel displacements. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'88)*, pages 742–746, IEEE Computer Society Press, Ann Arbor, Michigan.

- Kim, J., Kolmogorov, V., and Zabih, R. (2003). Visual correspondence using energy minimization and mutual information. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1033–1040, Nice, France.
- Kim, Y., Yu, J., Yu, X., and Lee, S. (2008). Line-art illustration of dynamic and specular surfaces. *ACM Transactions on Graphics*, 27(5).
- Kimura, R. et al.. (1999). A convolver-based real-time stereo machine (SAZAN). In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 457–463, Fort Collins.
- Kindermann, R. and Snell, J. L. (1980). *Markov Random Fields and Their Applications*. American Mathematical Society.
- Kirkpatrick, S., Gelatt, C. D. J., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.
- Kittler, J. and Föglein, J. (1984). Contextual classification of multispectral pixel data. *Image and Vision Computing*, 2, 13–19.
- Klinker, G. J. (1993). *A Physical Approach to Color Image Understanding*. A K Peters, Wellesley, Massachusetts.
- Koch, R., Pollefeys, M., and Van Gool, L. J. (2000). Realistic surface reconstruction of 3D scenes from uncalibrated image sequences. *Journal Visualization and Computer Animation*, 11, 115–127.
- Koenderink, J. J. (1990). *Solid Shape*. MIT Press, Cambridge, Massachusetts.
- Koethe, U. (2003). Integrated edge and junction detection with the boundary tensor. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 424–431, Nice, France.
- Kohli, P. and Torr, P. H. S. (2005). Efficiently solving dynamic markov random fields using graph cuts. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 922–929, Beijing, China.
- Kohli, P., Ladicky, L., and Torr, P. H. S. (2008). Robust higher order potentials for enforcing label consistency. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Kohli, P., Mudigonda, P., , and Torr, P. (2007). \mathcal{P}^3 & beyond: Solving energies with higher order cliques. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Kolev, K. et al.. (2007). Continuous global optimization in multiview 3d reconstruction. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 441–452, Springer-Verlag, EZhou, China.

- Kolmogorov, V. and Boykov, Y. (2005). What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 564–571, Beijing, China.
- Kolmogorov, V. *et al.*. (2006). Probabilistic fusion of stereo with color and contrast for bi-layer segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9), 1480–1492.
- Kolmogorov, V. and Zabih, R. (2001). Computing visual correspondence with occlusions using graph cuts. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 508–515, Vancouver, Canada.
- Kolmogorov, V. and Zabih, R. (2002). Multi-camera scene reconstruction via graph cuts. In *Seventh European Conference on Computer Vision (ECCV 2002)*, pages 82–96, Springer-Verlag, Copenhagen.
- Kolmogorov, V. and Zabih, R. (2004). What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2), 147–159.
- Kolomenkin, M., Shimshoni, I., and Tal, A. (2008). Demarcating curves for shape illustration. *ACM Transactions on Graphics*, 27(5).
- Komodakis, N. and Tziritas, G. (2007). Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *IEEE Transactions on Image Processing*, 16(11), 2649–2661.
- Komodakis, N., Tziritas, G., , and Paragios, N. (2007). Fast, approximately optimal solutions for single and dynamic mrfs. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Kopf, J., Cohen, M. F., Lischinski, D., and Uyttendaele, M. (2007a). Joint bilateral upsampling. *ACM Transactions on Graphics*, 26(3).
- Kopf, J., Uyttendaele, M., Deussen, O., and Cohen, M. F. (2007b). Capturing and viewing gigapixel images. *ACM Transactions on Graphics*, 26(3).
- Koutis, I. (2007). *Combinatorial and algebraic tools for optimal multilevel algorithms*. Ph.D. thesis, Carnegie Mellon University. Technical Report CMU-CS-07-131.
- Koutis, I. and Miller, G. L. (2007). A linear work, $o(n^{1/6})$ time, parallel algorithm for solving planar Laplacians. In *Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, Society for Industrial and Applied Mathematics, New Orleans.
- Koutis, I. and Miller, G. L. (2008). Graph partitioning into isolated, high conductance clusters: theory, computation and applications to preconditioning. In *Symposium on Parallel Algorithms and Architectures*, pages 137–145, Association for Computing Machinery, Munich.

- Košecká, J. and Zhang, W. (2005). Extraction, matching and pose recovery based on dominant rectangular structures. *Computer Vision and Image Understanding*, 100(3), 174–293.
- Kuglin, C. D. and Hines, D. C. (1975). The phase correlation image alignment method. In *IEEE 1975 Conference on Cybernetics and Society*, pages 163–165, New York.
- Kumar, M. P. and Torr, P. H. S. (2006). Fast memory-efficient generalized belief propagation. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 451–463, Springer.
- Kumar, M. P., Torr, P. H. S., and Zisserman, A. (2008). Learning layered motion segmentations of video. *International Journal of Computer Vision*, 76(3).
- Kumar, R., Anandan, P., and Hanna, K. (1994a). Direct recovery of shape from multiple views: A parallax based approach. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, pages 685–688, IEEE Computer Society Press, Jerusalem, Israel.
- Kumar, R., Anandan, P., and Hanna, K. (1994b). Shape recovery from multiple views: a parallax based approach. In *Image Understanding Workshop*, pages 947–955, Morgan Kaufmann Publishers, Monterey.
- Kumar, R., Anandan, P., Irani, M., Bergen, J., and Hanna, K. (1995). Representation of scenes from collections of images. In *IEEE Workshop on Representations of Visual Scenes*, pages 10–17, Cambridge, Massachusetts.
- Kumar, S. and Hebert, M. (2003a). Discriminative fields for modeling spatial dependencies in natural images. In *Advances in Neural Information Processing Systems*, MIT Press.
- Kumar, S. and Hebert, M. (2003b). Discriminative random fields: A discriminative framework for contextual interaction in classification. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1150–1157, Nice, France.
- Kumar, S. and Hebert, M. (2005). A hierarchical field framework for unified context-based classification. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1284–1291, Beijing, China.
- Kumar, S. and Hebert, M. (2006). Discriminative random fields. *International Journal of Computer Vision*, 68(2).
- Kutulakos, K. N. and Seitz, S. M. (1999). A theory of shape by space carving. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 307–314, Kerkyra, Greece.
- Kutulakos, K. N. and Seitz, S. M. (2000). A theory of shape by space carving. *International Journal of Computer Vision*, 38(3), 199–218.
- Kwatra, V. et al.. (2003). Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3), 277–286.

- Kybík, J. and Unser, M. (2003). Fast parametric elastic image registration. *IEEE Transactions on Image Processing*, 12(11), 1427–1442.
- Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*.
- Lagger, P., Salzmann, M., Lepetit, V., and Fua, P. (2008). 3d pose refinement from reflections. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Lai, S.-H. and Vemuri, B. C. (1997). Physically based adaptive preconditioning for early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6), 594–607.
- Langer, M. S. and Zucker, S. W. (1994). Shape from shading on a cloudy day. *Journal Optical Society America, A*, 11(2), 467–478.
- Lanitis, A., Taylor, C. J., and Cootes, T. F. (1995). A unified approach for coding and interpreting face images. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 368–373, Cambridge, Massachusetts.
- Lanitis, A., Taylor, C. J., and Cootes, T. F. (1997). Automatic interpretation and coding of face images using flexible models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), 742–756.
- Larlus, D. and Jurie, F. (2008). Combining appearance models and markov random fields for category level object segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Larson, G. W. (1998). LogLuv encoding for full-gamut, high-dynamic range images. *Journal of Graphics Tools*, 3(1), 15–31.
- Larson, G. W., Rushmeier, H., and Piatko, C. (1997). A visibility matching tone reproduction operator for high dynamic range scenes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4), 291–306.
- Laurentini, A. (1994). The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2), 150–162.
- Lavallée, S. and Szeliski, R. (1995). Recovering the position and orientation of free-form objects from image contours using 3-D distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(4), 378–390.
- Laveau, S. and Faugeras, O. D. (1994). 3-d scene representation as a collection of images. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, pages 689–691, IEEE Computer Society Press, Jerusalem, Israel.

- Lazebnik, S. *et al.*. (2005). A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8), 1265–1278.
- Lazebnik, S. *et al.*. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 2169–2176, New York City, NY.
- Le Gall, D. (1991). MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4), 46–58.
- Leclerc, Y. G. (1989). Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3(1), 73–102.
- Lee, M.-C. *et al.*. (1997). A layered video object coding system using sprite and affine motion model. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1), 130–145.
- Lee, Y. D., Terzopoulos, D., and Waters, K. (1995). Realistic facial modeling for animation. *Computer Graphics (SIGGRAPH'95)*, , 55–62.
- Lempitsky, V. and Boykov, Y. (2007). Global optimization for shape fitting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Lempitsky, V. and Ivanov, D. (2007). Seamless mosaicing of image-based texture maps. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Lempitsky, V. *et al.*. (2007). Logcut - efficient graph cut optimization for markov random fields. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Lempitsky, V., Blake, A., and Rother, C. (2008a). Image segmentation by branch-and-mincut. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 15–29, Springer-Verlag, Marseilles.
- Lempitsky, V., Roth, S., and Rother, C. (2008b). Flowfusion: Discrete-continuous optimization for optical flow estimation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Lengyel, J. and Snyder, J. (1997). Rendering with coherent layers. In *Computer Graphics Proceedings, Annual Conference Series*, pages 233–242, ACM SIGGRAPH, Proc. SIGGRAPH'97 (Los Angeles).
- Lepetit, V. and Fua, P. (2005). Monocular model-based 3d tracking of rigid objects. *Foundations and Trends in Computer Graphics and Computer Vision*, 1(1).
- Lepetit, V., Lagger, P., and Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 775–781, San Diego, CA.

- Levin, A., Acha, A. R., and Lischinski, D. (2008). Spectral matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10), 1699–1712.
- Levin, A., Lischinski, D., and Weiss, Y. (2004). Colorization using optimization. *ACM Transactions on Graphics*, 23(3), 689–694.
- Levin, A., Lischinski, D., and Weiss, Y. (2008). A closed form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2), 228–242.
- Levin, A., Zomet, A., and Weiss, Y. (2004). Separating reflections from a single image using local features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2004)*, pages 306–313, Washington, DC.
- Levin, A., Fergus, R., Durand, F., and Freeman, W. T. (2007). Image and depth from a conventional camera with a coded aperture. *ACM Transactions on Graphics*, 26(3).
- Levin, A., Zomet, A., Peleg, S., and Weiss, Y. (2004). Seamless image stitching in the gradient domain. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 377–389, Springer-Verlag, Prague.
- Levoy, M. (2006). Light fields and computational imaging. *Computer*, 39(8), 46–55.
- Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Computer Graphics Proceedings, Annual Conference Series*, pages 31–42, ACM SIGGRAPH, Proc. SIGGRAPH'96 (New Orleans).
- Levoy, M. *et al.*. (2000). The digital michelangelo project: 3d scanning of large statues. *Proceedings of SIGGRAPH 2000*, , 131–144.
- Leyvand, T., Cohen-Or, D., Dror, G., and Lischinski, D. (2008). Data-driven enhancement of facial attractiveness. *ACM Transactions on Graphics*, 27(3).
- Li, J., Shum, H., and Zhang, Y.-Q. (2000). On the compression of image based rendering scene. In *International Conference on Image Processing (ICIP-2000)*, pages 21–24, Vancouver.
- Li, S. (1995). *Markov Random Field Modeling in Computer Vision*. Springer-Verlag.
- Li, X., Wu, C., Zach, C., Lazebnik, S., and Frahm, J.-M. (2008). Modeling and recognition of landmark image collections using iconic scene graphs. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 427–440, Springer-Verlag, Marseilles.
- Li, Y., Shum, H.-Y., Tang, C.-K., and Szeliski, R. (2004a). Stereo reconstruction from multiperspective panoramas. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1), 44–62.
- Li, Y., Sun, J., Tang, C.-K., and Shum, H.-Y. (2004b). Lazy snapping. *ACM Transactions on Graphics*, 23(3), 303–308.

- Liang, L. *et al.*. (2001). Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3), 127–150.
- Lim, J. (1990). *Two-Dimensional Signal and Image Processing*. Prentice-Hall, Englewood, NJ.
- Lindeberg, T. (1990). Scale-space for discrete signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3), 234–254.
- Lindeberg, T. (1993). Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention. *International Journal of Computer Vision*, 11(3), 283–318.
- Lindeberg, T. (1994). Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2), 224–270.
- Lindeberg, T. (1998a). Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2), 116–154.
- Lindeberg, T. (1998b). Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2), 79–116.
- Lindeberg, T. and Gørding, J. (1997). Shape-adapted smoothing in estimation of 3-d shape cues from affine deformations of local 2-d brightness structure. *Image and Vision Computing*, 15(6), 415–434.
- Lischinski, D., Farbman, Z., Uyttendaele, M., and Szeliski, R. (2006a). Interactive local adjustment of tonal values. *ACM Transactions on Graphics*, 25(3), 646–653.
- Lischinski, D., Farbman, Z., Uyttendaele, M., and Szeliski, R. (2006b). Interactive local adjustment of tonal values. *ACM Transactions on Graphics*, 25(3), 646–653.
- Litvinov, A. and Schechner, Y. Y. (2005). Radiometric framework for image mosaicking. *Journal of the Optical Society of America A*, 22(5), 839–848.
- Litwinowicz, P. (1997). Processing images and video for an impressionist effect. In *Proceedings of SIGGRAPH 97*, pages 407–414.
- Litwinowicz, P. and Williams, L. (1994). Animating images with drawings. *Computer Graphics (SIGGRAPH'94)*, , 409–412.
- Liu, C., Szeliski, R., Kang, S. B., Zitnick, C. L., and Freeman, W. T. (2008). Automatic estimation and removal of noise from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2), 299–314.
- Longere, P. *et al.*. (2002). Perceptual assessment of demosaicing algorithm performance. *Proceedings of the IEEE*, 90(1), 123–132.

- Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293, 133–135.
- Loop, C. and Zhang, Z. (1999). Computing rectifying homographies for stereo vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 125–131, Fort Collins.
- Lorusso, A., Eggert, D., and Fisher, R. B. (1995). A comparison of four algorithms for estimating 3-d rigid transformations. In *British Machine Vision Conference (BMVC95)*, pages 237–246, Birmingham, England.
- Lourakis, M. I. A. and Argyros, A. A. (2004). *The Design and Implementation of a Generic Sparse Bundle Adjustment Software Package Based on the Levenberg-Marquardt Algorithm*. Technical Report 340, Institute of Computer Science - FORTH, Heraklion, Crete, Greece. Available from <http://www.ics.forth.gr/~lourakis/sba>.
- Lowe, D. G. (1988). Organization of smooth image curves at multiple scales. In *Second International Conference on Computer Vision (ICCV'88)*, pages 558–567, IEEE Computer Society Press, Tampa.
- Lowe, D. G. (1989). Organization of smooth image curves at multiple scales. *International Journal of Computer Vision*, 3(2), 119–130.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 1150–1157, Kerkyra, Greece.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 91–110.
- Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 674–679, Vancouver.
- Luong, Q.-T. and Faugeras, O. D. (1997). Self-calibration of a moving camera from point correspondences and fundamental matrices. *International Journal of Computer Vision*, 22(3), 261–289.
- Ma, Y., Derksen, H., Hong, W., and Wright, J. (2007). Segmentation of multivariate mixed data via lossy data coding and compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(9), 1546–1562.
- MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK.
- Magnor, M. and Girod, B. (2000). Data compression for light-field rendering. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(3), 338–343.

- Maire, M., Arbelaez, P., Fowlkes, C., and Malik, J. (2008). Using contours to detect and localize junctions in natural images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Maitre, M., Shinagawa, Y., and Do, M. N. (2008). Symmetric multi-view stereo reconstruction from planar camera arrays. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Maji, S., Berg, A., and Malik, J. (2008). Classification using intersection kernel support vector machines is efficient. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Malik, J. and Rosenholtz, R. (1997). Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision*, 23(2), 149–168.
- Malik, J., Belongie, S., Leung, T., and Shi, J. (2001). Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, 43(1), 7–27.
- Malisiewicz, T. and Efros, A. A. (2008). Recognition by association via learning per-exemplar distances. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Malladi, R., Sethian, J. A., and Vemuri, B. C. (1995). Shape modeling with front propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2), 158–176.
- Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(7), 674–693.
- Malvar, H. S. (1990). Lapped transforms for efficient transform/subband coding. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(6), 969–978.
- Malvar, H. S. (1998). Biorthogonal and nonuniform lapped transforms for transform coding with reduced blocking and ringing artifacts. *IEEE Transactions on Signal Processing*, 46(4), 1043–1053.
- Malvar, H. S. (2000). Fast progressive image coding without wavelets. In *IEEE Data Compression Conference*, pages 243–252, IEEE Computer Society Press, Snowbird, UT.
- Mancini, T. A. and Wolff, L. B. (1992). 3 d shape and light source location from depth and reflectance. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'92)*, pages 707–709, IEEE Computer Society Press, Champaign, Illinois.
- Mann, S. and Picard, R. W. (1994). Virtual bellows: Constructing high-quality images from video. In *First IEEE International Conference on Image Processing (ICIP-94)*, pages 363–367, Austin.

- Mann, S. and Picard, R. W. (1995). On being ‘undigital’ with digital cameras: Extending dynamic range by combining differently exposed pictures. In *IS&T’s 48th Annual Conference*, pages 422–428, Society for Imaging Science and Technology, Washington, D. C.
- Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, San Francisco.
- Marr, D. and Poggio, T. (1976). Cooperative computation of stereo disparity. *Science*, 194, 283–287.
- Marr, D. C. and Poggio, T. (1979). A computational theory of human stereo vision. *Proceedings of the Royal Society of London, B* 204, 301–328.
- Marroquin, J., Mitter, S., and Poggio, T. (1985). Probabilistic solution of ill-posed problems in computational vision. In *Image Understanding Workshop*, pages 293–309, Science Applications International Corporation, Miami Beach.
- Marroquin, J., Mitter, S., and Poggio, T. (1987). Probabilistic solution of ill-posed problems in computational vision. *Journal of the American Statistical Association*, 82(397), 76–89.
- Marroquin, J. L. (1983). *Design of Cooperative Networks*. Working Paper 253, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.
- Martin, D., Fowlkes, C., and Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5), 530–549.
- Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 416–423, Vancouver, Canada.
- Massey, M. and Bender, W. (1996). Salient stills: Process and practice. *IBM Systems Journal*, 35(3&4), 557–573.
- Matas, J. *et al.*. (2004). Robust wide baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10), 761–767.
- Matsushita, Y. and Lin, S. (2007). Radiometric calibration from noise distributions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Matsushita, Y. *et al.*. (2006). Full-frame video stabilization with motion inpainting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7), 1150–1163.
- Matthews, I. and Baker, S. (2004). Active appearance models revisited. *International Journal of Computer Vision*, 60(2), 135–164.

- Matthews, I. *et al.*. (2007). 2D vs. 3D deformable face models: Representational power, construction, and real-time fitting. *International Journal of Computer Vision*, 75(1), 93–113.
- Matthies, L. H., Szeliski, R., and Kanade, T. (1989). Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, 3, 209–236.
- Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., and McMillan, L. (2000). Image-based visual hulls. *Proceedings of SIGGRAPH 2000*, , 369–374. ISBN 1-58113-208-5.
- Maybeck, P. S. (1982). *Stochastic Models, Estimation, and Control*. Volume 2, Academic Press, New York.
- Mayhew, J. E. W. and Frisby, J. P. (1981). Psychophysical and computational studies towards a theory of human stereopsis. *Artificial Intelligence*, 17(1-3), 349–408.
- McLean, G. F. and Kotturi, D. (1995). Vanishing point detection by line clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11), 1090–1095.
- McGuire, M., Matusik, W., Pfister, H., Hughes, J. F., , and Durand, F. (2005). Defocus video matting. *ACM Transactions on Graphics*, 24(3), 567–576.
- McInerney, T. and Terzopoulos, D. (1993). A finite element model for 3D shape reconstruction and nonrigid motion tracking. In *Fourth International Conference on Computer Vision (ICCV'93)*, pages 518–523, IEEE Computer Society Press, Berlin, Germany.
- McLauchlan, P. F. (2000). A batch/recursive algorithm for 3D scene reconstruction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 738–743, Hilton Head Island.
- McLauchlan, P. F. and Jaenicke, A. (2002). Image mosaicing using sequential bundle adjustment. *Image and Vision Computing*, 20(9-10), 751–759.
- McLauchlan, P. F. and Murray, D. W. (1995). A unifying framework for structure and motion recovery from image sequences. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 230–236314–320, Cambridge, Massachusetts.
- McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An image-based rendering system. *Computer Graphics (SIGGRAPH'95)*, , 39–46.
- McMillan, L. and Gortler, S. (1999). Image-based rendering: A new interface between computer vision and computer graphics. *Computer Graphics*, 33(4), 61–64.
- Meehan, J. (1990). *Panoramic Photography*. Watson-Guptill.

- Meilă, M. and Shi, J. (2001). A random walks view of spectral segmentation. In Richardson, T. and Jaakkola, T., editors, *Workshop on Artificial Intelligence and Statistics*, pages 177–182, Society for Artificial Intelligence and Statistics, Key West, FL.
- Meilă, M. and Shi, J. (2006). Learning segmentation with random walk. In *Advances in Neural Information Processing Systems*, pages 873–879, MIT Press.
- Meltzer, J. and Soatto, S. (2008). Edge descriptors for robust wide-baseline correspondence. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Mémin, E. and Pérez, P. (2002). Hierarchical estimation and segmentation of dense motion fields. *International Journal of Computer Vision*, 44(2), 129–155.
- Menet, S., Saint-Marc, P., and Medioni, G. (1990a). Active contour models: overview, implementation and applications. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 194–199, Los Angeles.
- Menet, S., Saint-Marc, P., and Medioni, G. (1990b). B-snakes: implementation and applications to stereo. In *Image Understanding Workshop*, pages 720–726, Morgan Kaufmann Publishers, Pittsburgh.
- Mertens, T., Kautz, J., and Reeth, F. V. (2007). Exposure fusion. In *Proceedings of Pacific Graphics 2007*, pages 382–390.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1087–1091.
- Meyer, Y. (1993). *Wavelets: Algorithms and Applications*. Society for Industrial and Applied Mathematics, Philadelphia.
- Mikolajczyk, K. et al.. (2005). A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1-2), 43–72.
- Mikolajczyk, K. and Schmid, C. (2004). Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1), 63–86.
- Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10), 1615–1630.
- Milgram, D. L. (1975). Computer methods for creating photomosaics. *IEEE Transactions on Computers*, C-24(11), 1113–1119.
- Milgram, D. L. (1977). Adaptive techniques for photomosaicking. *IEEE Transactions on Computers*, C-26(11), 1175–1180.

- Mitiche, A. and Bouthemy, P. (1996). Computation and analysis of image motion: A synopsis of current problems and methods. *International Journal of Computer Vision*, 19(1), 29–55.
- Mitsunaga, T. and Nayar, S. K. (1999). Radiometric self calibration. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 374–380, Fort Collins.
- Mičušík, B., Wildenauer, H., and Košecká, J. (2008). Detection and matching of rectilinear structures. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Moezzi, S. *et al.*. (1996). Reality modeling and visualization from multiple video sequences. *IEEE Computer Graphics and Applications*, 16(6), 58–63.
- Moghaddam, B. and Pentland, A. (1997). Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), 696–710.
- Moghaddam, B., Nastar, C., and Pentland, A. (1996). Bayesian face recognition using deformable intensity surfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 638–645, San Francisco.
- Möller, K. D. (1988). *Optics*. University Science Books, Mill Valley, CA.
- Moravec, H. (1977). Towards automatic visual obstacle avoidance. In *Fifth International Joint Conference on Artificial Intelligence (IJCAI'77)*, page 584, Cambridge, Massachusetts.
- Moravec, H. (1983). The stanford cart and the cmu rover. *Proceedings of the IEEE*, 71(7), 872–884.
- Mori, G. (2005). Guiding model search using segmentation. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1417–1423, Beijing, China.
- Mori, G. *et al.*. (2004). Recovering human body configurations: Combining segmentation and recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2004)*, pages 326–333, Washington, DC.
- Mori, M. (1970). The uncanny valley. *Energy*, 7(4), 33–35. <http://www.androidscience.com/theuncannyvalley/proceedings2005/uncannyvalley.html>.
- Morimoto, C. and Chellappa, R. (1997). Fast 3D stabilization and mosaic construction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 660–665, San Juan, Puerto Rico.
- Morita, T. and Kanade, T. (1997). A sequential factorization method for recovering shape and motion from image streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8), 858–867.

- Morris, D. D. and Kanade, T. (1998). A unified factorization algorithm for points, line segments and planes with uncertainty models. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 696–702, Bombay.
- Morrone, M. and Burr, D. (1988). Feature detection in human vision: A phase dependent energy model. *Proceedings of the Royal Society of London B*, 235, 221–245.
- Mortensen, E. N. (1999). Vision-assisted image editing. *Computer Graphics*, 33(4), 55–57.
- Mortensen, E. N. and Barrett, W. A. (1995). Intelligent scissors for image composition. *Proceedings of SIGGRAPH 95*, , 191–198. ISBN 0-201-84776-0. Held in Los Angeles, California.
- Mortensen, E. N. and Barrett, W. A. (1999). Toboggan-based intelligent scissors with a four parameter edge model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 452–458, Fort Collins.
- Mueller, P., Zeng, G., Wonka, P., and Gool, L. V. (2007). Image-based procedural modeling of facades. *ACM Transactions on Graphics*, 26(3).
- Mühlich, M. and Mester, R. (1998). The role of total least squares in motion analysis. In *Fifth European Conference on Computer Vision (ECCV'98)*, pages 305–321, Springer-Verlag, Freiburg, Germany.
- Mumford, D. and Shah, J. (1989). Optimal approximations by piecewise smooth functions and variational problems. *Comm. Pure Appl. Math.*, XLII(5), 577–685.
- Mundy, J. L. and Zisserman, A., editors. (1992). *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, Massachusetts.
- Murphy, E. P. (2005). *A Testing Procedure to Characterize Color and Spatial Quality of Digital Cameras Used to Image Cultural Heritage*. Master's thesis, Rochester Institute of Technology.
- Murray, R. M., Li, Z. X., and Sastry, S. S. (1994). *A Mathematical Introduction to Robotic Manipulation*. CRC Press.
- Mutch, J. and Lowe, D. (2006). Multiclass object recognition with sparse, localized features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 11–18, New York City, NY.
- Nagel, H. H. (1986). Image sequences - ten (octal) years - from phenomenology towards a theoretical foundation. In *Eighth International Conference on Pattern Recognition (ICPR'86)*, pages 1174–1185, IEEE Computer Society Press, Paris.
- Nagel, H.-H. and Enkelmann, W. (1986). An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(5), 565–593.

- Nakamura, Y., Matsuura, T., Satoh, K., and Ohta, Y. (1996). Occlusion detectable stereo - occlusion patterns in camera matrix. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 371–378, San Francisco.
- Nakao, T., Kashitani, A., and Kaneyoshi, A. (1998). Scanning a document with a small camera attached to a mouse. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 63–68, IEEE Computer Society, Princeton.
- Nalwa, V. S. (1987). Edge-detector resolution improvement by image interpolation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9*(3), 446–451.
- Nalwa, V. S. (1993). *A Guided Tour of Computer Vision*. Addison-Wesley, Reading, MA.
- Nalwa, V. S. and Binford, T. O. (1986). On detecting edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8*(6), 699–714.
- Narasimhan, S. G. and Nayar, S. K. (2005). Enhancing resolution along multiple imaging dimensions using assorted pixels. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(4), 518–530.
- Nayar, S., Watanabe, M., and Noguchi, M. (1995). Real-time focus range sensor. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 995–1001, Cambridge, Massachusetts.
- Nayar, S. K. (2006). Computational cameras: Redefining the image. *Computer, 39*(8), 30–38.
- Nayar, S. K. and Branzoi, V. (2003). Adaptive dynamic range imaging: Optical control of pixel exposures over space and time. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1168–1175, Nice, France.
- Nayar, S. K. and Mitsunaga, T. (2000). High dynamic range imaging: Spatially varying pixel exposures. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 472–479, Hilton Head Island.
- Nayar, S. K., Ikeuchi, K., and Kanade, T. (1991). Shape from interreflections. *International Journal of Computer Vision, 6*(3), 173–195.
- Negahdaripour, S. and Yu, C. H. (1993). A generalized brightness change model for computing optical flow. In *Fourth International Conference on Computer Vision (ICCV'93)*, pages 2–11, IEEE Computer Society Press, Berlin, Germany.
- Neilson, D. and Yang, Y.-H. (2008). Evaluation of constructable match cost measures for stereo correspondence using cluster ranking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Nene, S. and Nayar, S. K. (1997). A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*(9), 989–1003.

- Netravali, A. and Robbins, J. (1979). Motion-compensated television coding: Part 1. *Bell System Tech.*, 58(3), 631–670.
- Nevatia, R. and Binford, T. (1977). Description and recognition of curved objects. *Artificial Intelligence*, 8, 77–98.
- Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–854, MIT Press.
- Nielsen, M., Florack, L. M. J., and Deriche, R. (1997). Regularization, scale-space, and edge-detection filters. *Journal of Mathematical Imaging and Vision*, 7(4), 291–307.
- Nielson, G. M. (1993). Scattered data modeling. *IEEE Computer Graphics and Applications*, 13(1), 60–70.
- Nir, T., Bruckstein, A. M., and Kimmel, R. (2008). Over-parameterized variational optical flow. *International Journal of Computer Vision*, 76(2).
- Nishihara, H. K. (1984). Practical real-time imaging stereo matcher. *OptEng*, 23(5), 536–545.
- Nistér, D. (2003). Preemptive RANSAC for live structure and motion estimation. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 199–206, Nice, France.
- Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 2161–2168, New York City, NY.
- Nistér, D. and Stewénius, H. (2008). Linear time maximally stable extremal regions. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 183–196, Springer-Verlag, Marseilles.
- Nomura, Y., Zhang, L., and Nayar, S. K. (2007). Scene collages and flexible camera arrays. In *Eurographics Symposium on Rendering*.
- Ohlander, R., Price, K., and Reddy, D. R. (1978). Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing*, 8(3), 313–333.
- Ohta, Y. and Kanade, T. (1985). Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(2), 139–154.
- Okutomi, M. and Kanade, T. (1992). A locally adaptive window for signal matching. *International Journal of Computer Vision*, 7(2), 143–162.
- Okutomi, M. and Kanade, T. (1993). A multiple baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4), 353–363.

- Okutomi, M. and Kanade, T. (1994). A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9), 920–932.
- Oliensis, J. (2000). A critique of structure-from-motion algorithms. *Computer Vision and Image Understanding*, 80(2), 172–214.
- Oliensis, J. and Genc, Y. (2001). Fast and accurate algorithms for projective multi-image structure from motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6), 546–559.
- Oliensis, J. and Hartley, R. (2007). Iterative extensions of the sturm/triggs algorithm: Convergence and nonconvergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12), 2217–2233.
- Oliva, A. and Torralba, A. (2001). Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3), 145–175.
- Omer, I. and Werman, M. (2004). Using natural image properties as demosaicing hints. In *International Conference on Image Processing (ICIP 2004)*, pages 1665–1670, Singapore.
- OpenGL-ARB. (1997). *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.1*. Addison-Wesley, Reading, MA, 2nd edition.
- Oppenheim, A. V. and Schafer, A. S. (1996). *Signals and Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition.
- Oppenheim, A. V., Schafer, R. W., and Buck, J. R. (1999). *Discrete-Time Signal Processing*. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition.
- Osher, S. and Paragios, N., editors. (2003). *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer.
- Ott, M., Lewis, J. P., and Cox, I. J. (1993). Teleconferencing eye contact using a virtual camera. In *INTERACT'93 and CHI'93 conference companion on Human factors in computing systems*, pages 109–110, ACM Press, Amsterdam.
- Pagliaroni, D. W. (1991). Distance transforms: Properties and machine vision applications. *Graphical Models and Image Processing*, 54(1), 56–74.
- Pal, C. et al.. (2004). Probability models for high dynamic range imaging. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2004)*, pages 173–180, Washington, DC.
- Papenberg, N. et al.. (2006). Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision*, 67(2), 141–158.
- Paris, S. and Durand, F. (2006). A fast approximation of the bilateral filter using a signal processing approach. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 568–580, Springer.

- Paris, S. and Durand, F. (2007). A topological approach to hierarchical segmentation using mean shift. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Parke, F. I. and Waters, K. (1996). *Computer Facial Animation*. A K Peters, Wellesley, Massachusetts.
- Parker, J. A., Kenyon, R. V., and Troxel, D. E. (1983). Comparison of interpolating methods for image resampling. *IEEE Transactions on Medical Imaging, MI-2(1)*, 31–39.
- Pavlidis, T. (1997). *Structural Pattern Recognition*. Springer-Verlag, Berlin; New York.
- Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, Los Altos.
- Peleg, R. and Ben-Ezra, M. (1999). Stereo panorama with a single camera. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 395–401, Fort Collins.
- Peleg, R., Ben-Ezra, M., and Pritch, Y. (2001). Omnistereo: Panoramic stereo imaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(3)*, 279–290.
- Peleg, S. (1981). Elimination of seams from photomosaics. *Computer Vision, Graphics, and Image Processing, 16(1)*, 1206–1210.
- Peleg, S. and Herman, J. (1997). Panoramic mosaics by manifold projection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 338–343, San Juan, Puerto Rico.
- Peleg, S. and Rav-Acha, A. (2006). Lucas-Kanade without iterative warping. In *International Conference on Image Processing (ICIP-2006)*, pages 1097–1100, Atlanta.
- Peleg, S., Rousso, B., Rav-Acha, A., and Zomet, A. (2000). Mosaicing on adaptive manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(10)*, 1144–1154.
- Pentland, A., Moghaddam, B., and Starner, T. (1994). View-based and modular eigenspaces for face recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, IEEE Computer Society, Seattle.
- Pentland, A. P. (1984). Local shading analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6(2)*, 170–179.
- Pentland, A. P. (1986). Perceptual organization and the representation of natural form. *Artificial Intelligence, 28(3)*, 293–331.
- Pentland, A. P. (1994). Interpolation using wavelet bases. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(4)*, 410–414.

- Pérez, P., Blake, A., and Gangnet, M. (2001). JetStream: Probabilistic contour extraction with particles. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 524–531, Vancouver, Canada.
- Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Transactions on Graphics*, 22(3), 313–318.
- Perona, P. (1995). Deformable kernels for early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5), 488–499.
- Perona, P. and Malik, J. (1990a). Detecting and localizing edges composed of steps, peaks and roofs. In *Third International Conference on Computer Vision (ICCV'90)*, pages 52–57, IEEE Computer Society Press, Osaka, Japan.
- Perona, P. and Malik, J. (1990b). Scale space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7), 629–639.
- Petschnigg, G., Agrawala, M., Hoppe, H., Szeliski, R., Cohen, M., and Toyama, K. (2004). Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics*, 23(3), 664–672.
- Philbin, J., Chum, O., Sivic, J., Isard, M., and Zisserman, A. (2008). Lost in quantization: Improving particular object retrieval in large scale image databases. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Philbin, J., Chum, O., Isard, M., Sivic, J., , and Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Phong, B. T. (1975). Illumination for computer generated pictures. *Communications of the ACM*, 18(6), 311–317.
- Pighin, F., Salesin, D. H., and Szeliski, R. (1999). Resynthesizing facial animation through 3D model-based tracking. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 143–150, Kerkyra, Greece.
- Pighin, F., Hecker, J., Lischinski, D., Salesin, D. H., and Szeliski, R. (1998). Synthesizing realistic facial expressions from photographs. In *Computer Graphics (SIGGRAPH'98 Proceedings)*, pages 75–84, ACM SIGGRAPH, Orlando.
- Pilet, J., Lepetit, V., and Fua, P. (2008). Fast non-rigid surface detection, registration, and realistic augmentation. *International Journal of Computer Vision*, 76(2).
- Pinz, A. (2005). Object categorization. *Foundations and Trends in Computer Graphics and Computer Vision*, 1(3).

- Platel, B., Balmachnova, E., Florack, L., and ter Haar Romeny, B. (2006). Top-points as interest points for image matching. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 418–429, Springer.
- Platt, J. C. (2000). Optimal filtering for patterned displays. *IEEE Signal Processing Letters*, 7(7), 179–180.
- Pock, T., Zach, C., , and Bischof, H. (2007). Mumford-shah meets stereo: Integration of weak depth hypotheses. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Poelman, C. J. and Kanade, T. (1997). A paraperspective factorization method for shape and motion recovery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3), 206–218.
- Poggio, T. and Koch, C. (1985). Ill-posed problems in early vision: from computational theory to analogue networks. *Proceedings of the Royal Society of London, B* 226, 303–323.
- Poggio, T. *et al.*. (1988). The MIT vision machine. In *Image Understanding Workshop*, pages 177–198, Morgan Kaufmann Publishers, Boston.
- Poggio, T., Torre, V., and Koch, C. (1985). Computational vision and regularization theory. *Nature*, 317(6035), 314–319.
- Pollard, S. B., Mayhew, J. E. W., and Frisby, J. P. (1985). PMF: A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14, 449–470.
- Pollefeys, M. and Van Gool, L. (2002). From images to 3D models. *Communications of the ACM*, 45(7), 50–55.
- Pollefeys, M. *et al.*. (2008). Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision*, 78(2-3), 143–167.
- Ponce, J. *et al.*, editors. (2007). *Toward Category-Level Object Recognition*, Springer, New York.
- Porter, T. and Duff, T. (1984). Compositing digital images. *Computer Graphics (SIGGRAPH'84)*, 18(3), 253–259.
- Portilla, J., Strela, V., Wainwright, M., and Simoncelli, E. P. (2003). Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11), 1338–1351.
- Potetz, B. (2007). Efficient belief propagation for vision using linear constraint nodes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Potmesil, M. (1987). Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40, 1–29.

- Pratt, W. K. (2001). *Digital Image Processing*. John Wiley & Sons, 3rd edition.
- Prazdny, K. (1985). Detection of binocular disparities. *Biological Cybernetics*, 52, 93–99.
- Pritchett, P. and Zisserman, A. (1998). Wide baseline stereo matching. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 754–760, Bombay.
- Proesmans, M., Van Gool, L., and Defoort, F. (1998). Reading between the lines - a method for extracting dynamic 3D with texture. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 1081–1086, Bombay.
- Pulli, K. (1999). Multiview registration for large data sets. In *Second International Conference on 3D Digital Imaging and Modeling (3DIM'99)*, pages 160–168, Ottawa, Canada.
- Pulli, K. et al.. (1998). Acquisition and visualization of colored 3D objects. In *International Conference on Pattern Recognition (ICPR'98)*, pages 11–15.
- Quam, L. H. (1984). Hierarchical warp stereo. In *Image Understanding Workshop*, pages 149–155, Science Applications International Corporation, New Orleans.
- Quan, L. and Lan, Z. (1999). Linear N-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8), 774–780.
- Quan, L. and Mohr, R. (1989). Determining perspective structures using hierarchical Hough transform. *Pattern Recognition Letters*, 9(4), 279–286.
- Rademacher, P. and Bishop, G. (1998). Multiple-center-of-projection images. In *Computer Graphics Proceedings, Annual Conference Series*, pages 199–206, ACM SIGGRAPH, Proc. SIGGRAPH'98 (Orlando).
- Raskar, R. and Tumblin, J. (2009). *Computational Photography: Mastering New Techniques for Lenses, Lighting, and Sensors*. A K Peters, Wellesley, Massachusetts.
- Raskar, R., Tan, K.-H., Feris, R., Yu, J., and Turk, M. (2004). Non-photorealistic camera: Depth edge detection and stylized rendering using multi-flash imaging. *ACM Transactions on Graphics*, 23(3), 679–?
- Rav-Acha, A., Kohli, P., Fitzgibbon, A., and Rother, C. (2008). Unwrap mosaics: A new representation for video editing. *ACM Transactions on Graphics*, 27(3).
- Rav-Acha, A., Pritch, Y., Lischinski, D., and Peleg, S. (2005). Dynamosaics: Video mosaics with non-chronological time. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 58–65, San Diego, CA.
- Ray, S. F. (2002). *Applied Photographic Optics*. Focal Press, Oxford, 3rd edition.
- Rehg, J. and Kanade, T. (1994). Visual tracking of high dof articulated structures: an application to human hand tracking. In *Third European Conference on Computer Vision (ECCV'94)*, pages 35–46, Springer-Verlag, Stockholm, Sweden.

- Rehg, J. and Witkin, A. (1991). Visual tracking with deformation models. In *IEEE International Conference on Robotics and Automation*, pages 844–850, IEEE Computer Society Press, Sacramento.
- Reichenbach, S. E., Park, S. K., and Narayanswamy, R. (1991). Characterizing digital image acquisition devices. *Optical Engineering*, 30(2), 170–177.
- Reinhard, E., Stark, M., Shirley, P., and Ferwerda, J. (2002). Photographic tone reproduction for digital images. *ACM Transactions on Graphics (TOG)*, 21(3), 267–276.
- Reinhard, E., Ward, G., Pattanaik, S., andDebevec, P. (2005). *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann.
- Rhemann, C., Rother, C., and Gelautz, M. (2008a). Improving color modeling for alpha matting. In *British Machine Vision Conference (BMVC 2008)*, Leeds.
- Rhemann, C., Rother, C., Rav-Acha, A., and Sharp, T. (2008b). High resolution matting via interactive trimap segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Rioul, O. and Vetterli, M. (1991). Wavelets and signal processing. *IEEE Signal Processing Magazine*, , 14–38.
- Rioux, M. and Bird, T. (1993). White laser, synced scan. *IEEE Computer Graphics and Applications*, 13(3), 15–17.
- Riseman, E. M. and Arbib, M. A. (1977). Computational techniques in the visual segmentation of static scenes. *Computer Graphics and Image Processing*, 6(3), 221–276.
- Ritter, G. X. and Wilson, J. N. (2000). *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, Boca Raton, 2nd edition.
- Roberts, L. G. (1965). Machine perception of three-dimensional solids. In Tippett *et al.*, editors, *Optical and Electro-Optical Information Processing*, chapter 9, pages 159–197, MIT Press, Cambridge, Massachusetts.
- Robertson, D. P. and Cipolla, R. (2002). Building architectural models from many views using map constraints. In *Seventh European Conference on Computer Vision (ECCV 2002)*, pages 155–169, Springer-Verlag, Copenhagen.
- Roble, D. (1999). Vision in film and special effects. *Computer Graphics*, 33(4), 58–60.
- Rogez, G., Rihan, J., Ramalingam, S., Orrite, C., and Torr, P. H. S. (2008). Randomized trees for human pose detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.

- Román, A. and Lensch, H. P. A. (2006). Automatic multiperspective images. In *Eurographics Symposium on Rendering*, pages 83–92.
- Román, A., Garg, G., and Levoy, M. (2004). Interactive design of multi-perspective images for visualizing urban landscapes. In *IEEE Visualization 2004*, pages 537–544, Minneapolis.
- Rosenfeld, A. (1980). Quadtrees and pyramids for pattern recognition and image processing. In *Fifth International Conference on Pattern Recognition (ICPR'80)*, pages 802–809, IEEE Computer Society Press, Miami Beach.
- Rosenfeld, A., editor. (1984). *Multiresolution Image Processing and Analysis*, Springer-Verlag, New York.
- Rosenfeld, A. and Davis, L. S. (1979). Image segmentation and image models. *Proceedings of the IEEE*, 67(5), 764–772.
- Rosenfeld, A. and Kak, A. C. (1976). *Digital Picture Processing*. Academic Press, New York.
- Rosenfeld, A., Hummel, R. A., and Zucker, S. W. (1976). Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-6*, 420–433.
- Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 430–443, Springer.
- Roth, S. and Black, M. (2005). Fields of experts: A framework for learning image priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 860–867, San Diego, CA.
- Roth, S. and Black, M. J. (2007). Steerable random fields. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Rother, C. (2002). A new approach for vanishing point detection in architectural environments. *Image and Vision Computing*, 20(9-10), 647–656.
- Rother, C. (2003). Linear multi-view reconstruction of points, lines, planes and cameras using a reference plane. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1210–1217, Nice, France.
- Rother, C., Kolmogorov, V., and Blake, A. (2004). “GrabCut”—interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3), 309–314.
- Rother, C., Kolmogorov, V., Lempitsky, V., , and Szummer, M. (2007). Optimizing binary mrf's via extended roof duality. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.

- Rousseeuw, P. J. (1984). Least median of squares regression. *Journal of the American Statistical Association*, 79, 871–880.
- Rousseeuw, P. J. and Leroy, A. M. (1987). *Robust Regression and Outlier Detection*. Wiley, New York.
- Rousson, M. and Paragios, N. (2008). Prior knowledge, level set representations, and visual grouping. *International Journal of Computer Vision*, 76(3), 231–243.
- Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear gaussian models. *Neural Computation*, 11(2), 305–345.
- Rowland, D. A. and Perrett, D. I. (1995). Manipulating facial appearance through shape and color. *IEEE Computer Graphics and Applications*, 15(5), 70–76.
- Roy, S. and Cox, I. J. (1998). A maximum-flow formulation of the N-camera stereo correspondence problem. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 492–499, Bombay.
- Rusinkiewicz, S. and Levoy, M. (2000). Qsplat: A multiresolution point rendering system for large meshes. *Proceedings of SIGGRAPH 2000*, , 343–352.
- Russ, J. C. (2007). *The Image Processing Handbook*. CRC Press, Boca Raton, 5th edition.
- Russell, B. C. *et al.*. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3), 157–173.
- Ruzon, M. A. and Tomasi, C. (2000). Alpha estimation in natural images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 18–25, Hilton Head Island.
- Ryan, T. W., Gray, R. T., and Hunt, B. R. (1980). Prediction of correlation errors in stereo-pair images. *Optical Engineering*, 19(3), 312–322.
- Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. SIAM, second edition.
- Saint-Marc, P., Chen, J. S., and Medioni, G. (1991). Adaptive smoothing: A general tool for early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6), 514–529.
- Saito, H. and Kanade, T. (1999). Shape reconstruction in projective grid space from large number of images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 49–54, Fort Collins.
- Salzmann, M., Urtasun, R., and Fua, P. (2008). Local deformation models for monocular 3d shape recovery. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Samet, H. (1989). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts.

- Sander, P. T. and Zucker, S. W. (1990). Inferring surface trace and differential structure from 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9), 833–854.
- Sapiro, G. (2001). *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press.
- Sato, Y. and Ikeuchi, K. (1996). Reflectance analysis for 3D computer graphics model generation. *Graphical Models and Image Processing*, 58(5), 437–451.
- Sato, Y., Wheeler, M., and Ikeuchi, K. (1997). Object shape and reflectance modeling from observation. In *Computer Graphics Proceedings, Annual Conference Series*, pages 379–387, ACM SIGGRAPH, Proc. SIGGRAPH'97 (Los Angeles).
- Sawhney, H. S. (1994a). 3D geometry from planar parallax. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 929–934, IEEE Computer Society, Seattle.
- Sawhney, H. S. (1994b). Simplifying motion and structure analysis using planar parallax and image warping. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, pages 403–408, IEEE Computer Society Press, Jerusalem, Israel.
- Sawhney, H. S. and Ayer, S. (1996). Compact representation of videos through dominant multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8), 814–830.
- Sawhney, H. S. and Hanson, A. R. (1991). Identification and 3D description of ‘shallow’ environmental structure over a sequence of images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 179–185, IEEE Computer Society Press, Maui, Hawaii.
- Sawhney, H. S. and Kumar, R. (1997). True multi-image alignment and its application to mosaicing and lens distortion correction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 450–456, San Juan, Puerto Rico.
- Sawhney, H. S. and Kumar, R. (1999). True multi-image alignment and its application to mosaicing and lens distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3), 235–243.
- Sawhney, H. S. et al.. (1998). Videobrush: Experiences with consumer video mosaicing. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 56–62, IEEE Computer Society, Princeton.
- Schaffalitzky, F. and Zisserman, A. (2000). Planar grouping for automatic detection of vanishing lines and points. *Image and Vision Computing*, 18, 647–658.
- Schaffalitzky, F. and Zisserman, A. (2002). Multi-view matching for unordered image sets, or “How do I organize my holiday snaps?”. In *Seventh European Conference on Computer Vision (ECCV 2002)*, pages 414–431, Springer-Verlag, Copenhagen.

- Scharr, H., Black, M. J., and Haussecker, H. W. (2003). Image statistics and anisotropic diffusion. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 840–847, Nice, France.
- Scharstein, D. (1994). Matching images by comparing their gradient fields. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, pages 572–575, IEEE Computer Society Press, Jerusalem, Israel.
- Scharstein, D. (1999). *View Synthesis Using Stereo Vision*. Volume 1583 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag.
- Scharstein, D. and Pal, C. (2007). Learning conditional random fields for stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Scharstein, D. and Szeliski, R. (1998). Stereo matching with nonlinear diffusion. *International Journal of Computer Vision*, 28(2), 155–174.
- Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1), 7–42.
- Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2003)*, pages 195–202, Madison, WI.
- Schechner, Y. Y., Nayar, S. K., and Belhumeur, P. (2003). A theory of multiplexed illumination. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 808–815, Nice, France.
- Schindler, G., Brown, M., and Szeliski, R. (2007). City-scale location recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Schindler, G., Krishnamurthy, P., Lublinerman, R., Liu, Y., and Dellaert, F. (2008). Detecting and matching repeated patterns for automatic geo-tagging in urban environments. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Schmid, C. and Mohr, R. (1997). Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5), 530–534.
- Schmid, C. and Zisserman, A. (1997). Automatic line matching across views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 666–671, San Juan, Puerto Rico.
- Schmid, C., Mohr, R., and Bauckhage, C. (2000). Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2), 151–172.
- Schödl, A., Szeliski, R., Salesin, D. H., and Essa, I. (2000). Video textures. In *Computer Graphics (SIGGRAPH'2000 Proceedings)*, pages 489–498, ACM SIGGRAPH, New Orleans.

- Schoenemann, T. and Cremers, D. (2008). High resolution motion layer decomposition using dual-space graph cuts. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Schröder, P. and Sweldens, W. (1995). Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of SIGGRAPH 95*, pages 161–172.
- Scott, G. L. and Longuet-Higgins, H. C. (1990). Feature grouping by relocalization of eigenvectors of the proximity matrix. In *British Machine Vision Conference*, pages 103–108.
- Sederberg, T. W., Gao, P., Wang, G., and Mu, H. (1993). 2d shape blending: An intrinsic solution to the vertex path problem. In *Computer Graphics (Proceedings of SIGGRAPH 93)*, pages 15–18.
- Seitz, P. (1989). Using local orientation information as image primitive for robust object recognition. In *SPIE Visual Communications and Image Processing IV*, pages 1630–1639.
- Seitz, S. (2001). The space of all stereo images. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 26–33, Vancouver, Canada.
- Seitz, S. and Szeliski, R. (1999). Applications of computer vision to computer graphics. *Computer Graphics*, 33(4), 35–37. Guest Editors’ introduction to the Special Issue.
- Seitz, S., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’2006)*, pages 519–526, New York, NY.
- Seitz, S. M. and Anandan, P. (1999). Implicit representation and scene reconstruction from probability density functions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’99)*, pages 28–34, Fort Collins.
- Seitz, S. M. and Dyer, C. M. (1996). View morphing. In *Computer Graphics Proceedings, Annual Conference Series*, pages 21–30, ACM SIGGRAPH, Proc. SIGGRAPH’96 (New Orleans).
- Seitz, S. M. and Dyer, C. M. (1997). Photorealistic scene reconstruction by voxel coloring. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’97)*, pages 1067–1073, San Juan, Puerto Rico.
- Seitz, S. M. and Dyer, C. M. (1999). Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2), 151–173.
- Serra, J. P. (1982). *Image Analysis and Mathematical Morphology*. Academic Press, New York.
- Sethian, J. (1999). *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, 2nd edition.

- Shade, J., Gortler, S., He, L.-W., and Szeliski, R. (1998). Layered depth images. In *Computer Graphics (SIGGRAPH'98 Proceedings)*, pages 231–242, ACM SIGGRAPH, Orlando.
- Shade, J., Lischinski, D., Salesin, D., DeRose, T., and Snyder, J. (1996). Hierarchical images caching for accelerated walkthroughs of complex environments. In *Computer Graphics (SIGGRAPH'96) Proceedings*, pages 75–82, ACM SIGGRAPH, Proc. SIGGRAPH'96 (New Orleans).
- Shafer, S. A. (1985). Using color to separate reflection components. *COLOR Research and Applications*, 10(4), 210–218.
- Shafer, S. A., Healey, G., and Wolff, L. (1992). *Physics-Based Vision: Principles and Practice*. Jones & Bartlett, Cambridge, MA.
- Shafique, K. and Shah, M. (2005). A noniterative greedy algorithm for multiframe point correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(1), 51–65.
- Shah, J. (1993). A nonlinear diffusion model for discontinuous disparity and half-occlusion in stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 34–40, New York.
- Shakhnarovich, G., Darrell, T., and Indyk, P., editors. (2006). *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, MIT Press.
- Shakhnarovich, G., Viola, P., and Darrell, T. (2003). Fast pose estimation with parameter-sensitive hashing. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 750–757, Nice, France.
- Shan, Y., Liu, Z., and Zhang, Z. (2001). Model-based bundle adjustment with application to face modeling. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 644–641, Vancouver, Canada.
- Sharon, E., Galun, M., Sharon, D., Basri, R., and Brandt, A. (2006). Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442(7104), 810–813.
- Shashua, A. and Toelg, S. (1997). The quadric reference surface: Theory and applications. *International Journal of Computer Vision*, 23(2), 185–198.
- Shashua, A. and Wexler, Y. (2001). Q-warping: Direct computation of quadratic reference surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8), 920–925.
- Shaw, D. and Barnes, N. (2006). Perspective rectangle detection. In *Workshop on Applications of Computer Vision at ECCV'2006*, Springer-Verlag.
- Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Unpublished manuscript, available on author's homepage (<http://www.cs.berkeley.edu/~jrs/>). An earlier version appeared as a Carnegie Mellon University Technical Report, CMU-CS-94-125.

- Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 888–905.
- Shi, J. and Tomasi, C. (1994). Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, IEEE Computer Society, Seattle.
- Shimizu, M. and Okutomi, M. (2001). Precise sub-pixel estimation on area-based matching. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 90–97, Vancouver, Canada.
- Shirley, P. (2005). *Fundamentals of Computer Graphics*. A K Peters, Wellesley, Massachusetts, second edition.
- Shoemake, K. (1985). Animating rotation with quaternion curves. *Computer Graphics (SIGGRAPH'85)*, 19(3), 245–254.
- Shotton, J. *et al.*. (2008a). Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling appearance, shape and context. *International Journal of Computer Vision*, .
- Shotton, J., Johnson, M., and Cipolla, R. (2008b). Semantic texton forests for image categorization and segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). *TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 1–15, Springer.
- Shufelt, J. (1999). Performance evaluation and analysis of vanishing point detection techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3), 282–288.
- Shum, H.-Y. and He, L.-W. (1999). Rendering with concentric mosaics. In *SIGGRAPH'99*, pages 299–306, ACM SIGGRAPH, Los Angeles.
- Shum, H.-Y. *et al.*. (1999). Omnivergenet stereo. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 22–29, Greece.
- Shum, H.-Y. *et al.*. (2002). Rendering by manifold hopping. *International Journal of Computer Vision*, 50(2), 185–201.
- Shum, H.-Y. and Szeliski, R. (1997). *Panoramic Image Mosaicing*. Technical Report MSR-TR-97-23, Microsoft Research.
- Shum, H.-Y. and Szeliski, R. (1999). Stereo reconstruction from multiperspective panoramas. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 14–21, Kerkyra, Greece.

- Shum, H.-Y. and Szeliski, R. (2000). Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision*, 36(2), 101–130. Erratum published July 2002, 48(2):151-152.
- Shum, H.-Y., Chan, S.-C., and Kang, S. B. (2007). *Image-Based Rendering*. Springer, New York, NY.
- Shum, H.-Y., Han, M., and Szeliski, R. (1998). Interactive construction of 3D models from panoramic mosaics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 427–433, Santa Barbara.
- Sidenbladh, H. and Black, M. J. (2003). Learning the statistics of people in images and video. *International Journal of Computer Vision*, 54(1), 189–209.
- Sillion, F. and Puech, C. (1994). *Radiosity and Global Illumination*. Morgan Kaufmann.
- Simard, P. Y. *et al.* (1998). Boxlets: a fast convolution algorithm for signal processing and neural networks. In Kearns, M. *et al.*, editors, *Advances in Neural Information Processing Systems 13*, pages 571–577, MIT Press.
- Simoncelli, E. P. and Adelson, E. H. (1990a). Non-separable extensions of quadrature mirror filters to multiple dimensions. *Proceedings of the IEEE*, 78(4), 652–664.
- Simoncelli, E. P. and Adelson, E. H. (1990b). Subband transforms. In Woods, J., editor, *Subband Coding*, pages 143–191, Kluwer Academic Press, Norwell, MA.
- Simoncelli, E. P., Adelson, E. H., and Heeger, D. J. (1991). Probability distributions of optic flow. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 310–315, IEEE Computer Society Press, Maui, Hawaii.
- Simoncelli, E. P., Freeman, W. T., Adelson, E. H., and Heeger, D. J. (1992). Shiftable multiscale transforms. *IEEE Transactions on Information Theory*, 38(3), 587–607.
- Singaraju, D., Grady, L., and Vidal, R. (2008). Interactive image segmentation via minimization of quadratic energies on directed graphs. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Sinha, S. N., Steedly, D., Szeliski, R., Agrawala, M., and Pollefeys, M. (2008). Interactive 3D architectural modeling from unordered photo collections. *ACM Transactions on Graphics*, 27(5).
- Sivic, J. and Zisserman, A. (2003). Video google: A text retrieval approach to object matching in videos. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1470–1477, Nice, France.
- Sivic, J., Zitnick, C. L., and Szeliski, R. (2006). Finding people in repeated shots of the same scene. In *British Machine Vision Conference (BMVC 2006)*, pages 909–918, Springer-Verlag, Edinburgh.

- Sivic, J., Russell, B., Zisserman, A., Freeman, W. T., and Efros, A. A. (2008). Unsupervised discovery of visual object class hierarchies. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Slama, C. C., editor. (1980). *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, Virginia, fourth edition.
- Smelyanskiy, V. N., Cheeseman, P., Maluf, D. A., and Morris, R. D. (2000). Bayesian super-resolved surface reconstruction from images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 375–382, Hilton Head Island.
- Smith, A. R. and Blinn, J. F. (1996). Blue screen matting. In *Computer Graphics Proceedings, Annual Conference Series*, pages 259–268, ACM SIGGRAPH, Proc. SIGGRAPH'96 (New Orleans).
- Smith, S. M. and Brady, J. M. (1997). SUSAN—a new approach to low level image processing. *International Journal of Computer Vision*, 23(1), 45–78.
- Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring photo collections in 3D. *ACM Transactions on Graphics*, 25(3), 835–846.
- Snavely, N., Seitz, S. M., and Szeliski, R. (2008a). Skeletal graphs for efficient structure from motion. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Snavely, N., Garg, R., Seitz, S. M., and Szeliski, R. (2008b). Finding paths through the world's photos. *ACM Transactions on Graphics*, 27(3).
- Soatto, S., Doretto, G., and Wu, Y. N. (2001). Dynamic textures. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 439–446, Vancouver, Canada.
- Soatto, S., Yezzi, A. J., and Jin, H. (2003). Tales of shape and radiance in multiview stereo. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 974–981, Nice, France.
- Soille, P. (2006). Morphological image compositing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5), 673–683.
- Solina, F. and Bajcsy, R. (1990). Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2), 131–147.
- Sorenson, H. W. (1980). *Parameter Estimation, Principles and Problems*. Marcel Dekker, New York.
- Soucy, M. and Laurendeau, D. (1992). Multi-resolution surface modeling from multiple range views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'92)*, pages 348–353, IEEE Computer Society Press, Champaign, Illinois.

- Srinivasan, S. *et al.*. (2005). Electronic image stabilization and mosaicking algorithms. In Bovik, A., editor, *Handbook of Image and Video Processing*, Academic Press.
- Sriwasan, P., Liang, P., and Hackwood, S. (1990). Computational geometric methods in volumetric intersections for 3D reconstruction. *Pattern Recognition*, 23(8), 843–857.
- Steedly, D. and Essa, I. (2001). Propagation of innovative information in non-linear least-squares structure from motion. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 223–229, Vancouver, Canada.
- Steedly, D. *et al.*. (2005). Efficiently registering video into panoramic mosaics. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1300–1307, Beijing, China.
- Steedly, D., Essa, I., and Dellaert, F. (2003). Spectral partitioning for structure from motion. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 996–1003, Nice, France.
- Steele, R. and Jaynes, C. (2005). Feature uncertainty arising from covariant image noise. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 1063–1070, San Diego, CA.
- Stein, A. *et al.*. (2007). Learning to extract object boundaries using motion cues. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Stein, G. (1995). Accurate internal camera calibration using rotation, with analysis of sources of error. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 230–236, Cambridge, Massachusetts.
- Stein, G. (1997). Lens distortion calibration using point correspondences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 602–608, San Juan, Puerto Rico.
- Stewart, C. V. (1999). Robust parameter estimation in computer vision. *SIAM Reviews*, 41(3), 513–537.
- Stiller, C. and Konrad, J. (1999). Estimating motion in image sequences: A tutorial on modeling and computation of 2d motion. *IEEE Signal Processing Magazine*, 16(4), 70–91.
- Stollnitz, E. J., DeRose, T. D., and Salesin, D. H. (1996). *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, San Francisco.
- Strang, G. (1988). *Linear Algebra and its Applications*. Harcourt, Brace, Jovanovich, Publishers, San Diego, 3rd edition.
- Strang, G. (1989). Wavelets and dilation equations: A brief introduction. *SIAM Reviews*, 31(4), 614–627.
- Strecha, C., Tuytelaars, T., and Gool, L. V. (2003). Dense matching of multiple wide-baseline views. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 1194–1201, Nice, France.

- Sturm, P. and Ramalingam, S. (2004). A generic concept for camera calibration. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 1–13, Springer-Verlag, Prague.
- Sudderth, E. B., Torralba, A., Freeman, W. T., and Willsky, A. S. (2005). Learning hierarchical models of scenes, objects, and parts. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1331–1338, Beijing, China.
- Sullivan, S. and Ponce, J. (1998). Automatic model construction and pose estimation from photographs using triangular splines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10), 1091–1096.
- Sun, D., Roth, S., Lewis, J., and Black, M. J. (2008). Learning optical flow. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 83–97, Springer-Verlag, Marseilles.
- Sun, J. *et al.*. (2004). Image completion with structure propagation. *ACM Transactions on Graphics*, 24(3), 861–868.
- Sun, J., Zheng, N., and Shum, H. (2003). Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7), 787–800.
- Sun, J., Jia, J., Tang, C.-K., and Shum, H.-Y. (2004). Poisson matting. *ACM Transactions on Graphics*, 23(3), 315–321.
- Sun, J., Li, Y., Kang, S. B., and Shum, H.-Y. (2006). Flash matting. *ACM Transactions on Graphics*, 25(3), 772–778.
- Swain, M. J. and Ballard, D. H. (1991). Color indexing. *International Journal of Computer Vision*, 7(1), 11–32.
- Swaminathan, R., Kang, S. B., Szeliski, R., Criminisi, A., and Nayar, S. K. (2002). On the motion and appearance of specularities in image sequences. In *Seventh European Conference on Computer Vision (ECCV 2002)*, pages 508–523, Springer-Verlag, Copenhagen.
- Sweldens, W. (1996). Wavelets and the lifting scheme: A 5 minute tour. *Z. Angew. Math. Mech.*, 76 (Suppl. 2), 41–44.
- Sweldens, W. (1997). The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2), 511–546.
- Swendsen, R. H. and Wang, J.-S. (1987). Nonuniversal critical dynamics in Monte Carlo simulations. *Physical Review Letters*, 58(2), 86–88.
- Szeliski, R. (1986). *Cooperative Algorithms for Solving Random-Dot Stereograms*. Technical Report CMU-CS-86-133, Computer Science Department, Carnegie Mellon University.

- Szeliski, R. (1990a). Bayesian modeling of uncertainty in low-level vision. *International Journal of Computer Vision*, 5(3), 271–301.
- Szeliski, R. (1990b). Fast surface interpolation using hierarchical basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6), 513–528.
- Szeliski, R. (1991a). Fast shape from shading. *CVGIP: Image Understanding*, 53(2), 129–153.
- Szeliski, R. (1991b). Shape from rotation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 625–630, IEEE Computer Society Press, Maui, Hawaii.
- Szeliski, R. (1993). Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1), 23–32.
- Szeliski, R. (1994). Image mosaicing for tele-reality applications. In *IEEE Workshop on Applications of Computer Vision (WACV'94)*, pages 44–53, IEEE Computer Society, Sarasota.
- Szeliski, R. (1996). Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2), 22–30.
- Szeliski, R. (1999). A multi-view approach to motion and stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, pages 157–163, Fort Collins.
- Szeliski, R. (2006a). Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Computer Vision*, 2(1).
- Szeliski, R. (2006b). Locally adapted hierarchical basis preconditioning. *ACM Transactions on Graphics*, 25(3), 1135–1143.
- Szeliski, R. and Coughlan, J. (1997). Spline-based image registration. *International Journal of Computer Vision*, 22(3), 199–218.
- Szeliski, R. and Golland, P. (1999). Stereo matching with transparency and matting. *International Journal of Computer Vision*, 32(1), 45–61. Special Issue for Marr Prize papers.
- Szeliski, R. and Hinton, G. (1985). Solving random-dot stereograms using the heat equation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'85)*, pages 284–288, IEEE Computer Society Press, San Francisco.
- Szeliski, R. and Ito, M. R. (1986). New Hermite cubic interpolator for two-dimensional curve generation. *IEE Proceedings E*, 133(6), 341–347.
- Szeliski, R. and Kang, S. B. (1994). Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5(1), 10–28.

- Szeliski, R. and Kang, S. B. (1995). Direct methods for visual scene reconstruction. In *IEEE Workshop on Representations of Visual Scenes*, pages 26–33, Cambridge, Massachusetts.
- Szeliski, R. and Kang, S. B. (1997). Shape ambiguities in structure from motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5), 506–512.
- Szeliski, R. and Lavallee, S. (1996). Matching 3-D anatomical surfaces with non-rigid deformations using octree-splines. *International Journal of Computer Vision*, 18(2), 171–186.
- Szeliski, R. and Scharstein, D. (2004). Sampling the disparity space image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3), 419–425.
- Szeliski, R. and Shum, H.-Y. (1996). Motion estimation with quadtree splines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(12), 1199–1210.
- Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and texture-mapped models. *Computer Graphics (SIGGRAPH'97 Proceedings)*, , 251–258.
- Szeliski, R. and Tonnesen, D. (1992). Surface modeling with oriented particle systems. *Computer Graphics (SIGGRAPH'92)*, 26(2), 185–194.
- Szeliski, R. and Torr, P. (1998). Geometrically constrained structure from motion: Points on planes. In Koch, R. and Van Gool, L., editors, *European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE)*, pages 171–186, Freiburg, Germany.
- Szeliski, R. and Weiss, R. (1998). Robust shape recovery from occluding contours using a linear smoother. *International Journal of Computer Vision*, 28(1), 27–44.
- Szeliski, R., Avidan, S., and Anandan, P. (2000). Layer extraction from multiple images containing reflections and transparency. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2000)*, pages 246–253, Hilton Head Island.
- Szeliski, R., Tonnesen, D., and Terzopoulos, D. (1993a). Curvature and continuity control in particle-based surface models. In *SPIE Vol. 2031 Geometric Methods in Computer Vision II*, pages 172–181, Society of Photo-Optical Instrumentation Engineers, San Diego.
- Szeliski, R., Tonnesen, D., and Terzopoulos, D. (1993b). Modeling surfaces of arbitrary topology with dynamic particles. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 82–87, New York.
- Szeliski, R., Uyttendaele, M., and Steedly, D. (2008a). *Fast Poisson Blending using Multi-Splines*. Technical Report MSR-TR-2008-58, Microsoft Research.
- Szeliski, R., Winder, S., and Uyttendaele, M. (2008b). *High-quality multi-pass image resampling*. Technical Report MSR-TR-2008-???, Microsoft Research.

- Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2008c). A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6), 1068–1080.
- Szeliski, R. S. (1989). *Bayesian Modeling of Uncertainty in Low-Level Vision*. Kluwer Academic Publishers, Boston.
- Szummer, M. and Picard, R. W. (1996). Temporal texture modeling. In *IEEE International Conference on Image Processing (ICIP-96)*, pages 823–826, Lausanne.
- Tabb, M. and Ahuja, N. (1997). Multiscale image segmentation by integrated edge and region detection. *IEEE Transactions on Image Processing*, 6(5), 642–655.
- Taguchi, Y., Wilburn, B., and Zitnick, C. L. (2008). Stereo reconstruction with mixed pixels using adaptive over-segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Tanaka, M. and Okutomi, M. (2008). Locally adaptive learning for translation-variant mrf image priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Tao, H., Sawhney, H., and Kumar, R. (2001). A global matching framework for stereo computation. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 532–539, Vancouver, Canada.
- Tappen, M. (2007). Utilizing variational optimization to learn markov random fields. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Tappen, M., Liu, C., Freeman, W., , and Adelson, E. (2007). Learning gaussian conditional random fields for low-level vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Tappen, M. F. and Freeman, W. T. (2003). Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 900–907, Nice, France.
- Tappen, M. F. *et al.*. (2003). Exploiting the sparse derivative prior for super-resolution and image de-mosaicing. In *Third International Workshop on Statistical and Computational Theories of Vision*, Nice, France.
- Tappen, M. F., Freeman, W. T., and Adelson, E. H. (2005). Recovering intrinsic images from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9), 1459–1472.
- Tardif, J.-P. *et al.*. (2006). Self-calibration of a general radially symmetric distortion model. In *Ninth European Conference on Computer Vision (ECCV 2006)*, pages 186–199, Springer-Verlag, Graz.

- Taubin, G. (1995). Curve and surface smoothing without shrinkage. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 852–857, Cambridge, Massachusetts.
- Taubman, D. S. and Marcellin, M. W. (2002). Jpeg2000: standard for interactive imaging. *Proceedings of the IEEE*, 90(8), 1336–1357.
- Taylor, C. J. (2003). Surface reconstruction from feature based stereo. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 184–190, Nice, France.
- Taylor, C. J., Debevec, P. E., and Malik, J. (1996). Reconstructing polyhedral models of architectural scenes from photographs. In *Fourth European Conference on Computer Vision (ECCV'96)*, pages 659–668, Springer-Verlag, Cambridge, England.
- Taylor, C. J., Kriegman, D. J., and Anandan, P. (1991). Structure and motion in two dimensions from multiple images: A least squares approach. In *IEEE Workshop on Visual Motion*, pages 242–248, IEEE Computer Society Press, Princeton, New Jersey.
- Tekalp, M. (1995). *Digital Video Processing*. Prentice Hall, Upper Saddle River, NJ.
- Telea, A. (2004). An image inpainting technique based on fast marching method. *Journal of Graphics Tools*, 9(1), 23–34.
- Teodosio, L. and Bender, W. (1993). Salient video stills: Content and context preserved. In *ACM Multimedia 93*, pages 39–46, Anaheim, California.
- Terzopoulos, D. (1983). Multilevel computational processes for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 24, 52–96.
- Terzopoulos, D. (1986). Regularization of inverse visual problems involving discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(4)*, 413–424.
- Terzopoulos, D. (1988). The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-10(4)*, 417–438.
- Terzopoulos, D. (1999). Visual modeling for computer animation: Graphics with a vision. *Computer Graphics*, 33(4), 42–45.
- Terzopoulos, D. and Fleischer, K. (1988). Deformable models. *The Visual Computer*, 4(6), 306–331.
- Terzopoulos, D. and Metaxas, D. (1991). Dynamic 3D models with local and global deformations: Deformable superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7), 703–714.
- Terzopoulos, D. and Szeliski, R. (1992). Tracking with Kalman snakes. In Blake, A. and Yuille, A. L., editors, *Active Vision*, pages 3–20, MIT Press, Cambridge, Massachusetts.

- Terzopoulos, D. and Waters, K. (1990). Analysis of facial images using physical and anatomical models. In *Third International Conference on Computer Vision (ICCV'90)*, pages 727–732, Osaka, Japan.
- Terzopoulos, D. and Witkin, A. (1988). Physically-based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6), 41–51.
- Terzopoulos, D., Witkin, A., and Kass, M. (1987). Symmetry-seeking models and 3D object reconstruction. *International Journal of Computer Vision*, 1(3), 211–221.
- Terzopoulos, D., Witkin, A., and Kass, M. (1988). Constraints on deformable models: Recovering 3D shape and nonrigid motion. *Artificial Intelligence*, 36(1), 91–123.
- Thayananthan, A., Iwasaki, M., and Cipolla, R. (2008). Principled fusion of high-level model and low-level cues for motion segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Tian, Q. and Huhns, M. N. (1986). Algorithms for subpixel registration. *Computer Vision, Graphics, and Image Processing*, 35, 220–233.
- Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solutions of Ill-Posed Problems*. V. H. Winston, Washington, D. C.
- Tolliver, D. and Miller, G. (2006). Graph partitioning by spectral rounding: Applications in image segmentation and clustering. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 1053–1060, New York City, NY.
- Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 9(2), 137–154.
- Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 839–846, Bombay.
- Tombari, F., Mattoccia, S., Di Stefano, L., and Addimanda, E. (2008). Classification and evaluation of cost aggregation methods for stereo correspondence. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Tommasini, T. *et al.*. (1998). Making good features track better. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 178–183, IEEE Computer Society, Santa Barbara.
- Torborg, J. and Kajiya, J. T. (1996). Talisman: Commodity realtime 3D graphics for the PC. In *Computer Graphics Proceedings, Annual Conference Series*, pages 353–363, ACM SIGGRAPH, Proc. SIGGRAPH'96 (New Orleans).

- Torr, P. H. S. and Murray, D. W. (1997). The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3), 271–300.
- Torr, P. H. S., Szeliski, R., and Anandan, P. (1999). An integrated Bayesian approach to layer extraction from image sequences. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 983–990, Kerkyra, Greece.
- Torr, P. H. S., Szeliski, R., and Anandan, P. (2001). An integrated Bayesian approach to layer extraction from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3), 297–303.
- Torralba, A. (2003). Contextual priming for object detection. *International Journal of Computer Vision*, 53(2), 169–191.
- Torralba, A. (2007). Classifier-based methods. In *Recognizing and Learning Object Categories*, page CVPR 2007 Short Course on Recognizing and Learning Object Categories. <http://people.csail.mit.edu/torralba/shortCourseRLOC/>.
- Torralba, A., Murphy, K. P., Freeman, W. T., and Rubin, M. A. (2003). Context-based vision system for place and object recognition. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 273–280, Nice, France.
- Torrance, K. E. and Sparrow, E. M. (1967). Theory for off-specular reflection from roughened surfaces. *Journal of the Optical Society of America A*, 57(9), 1105–1114.
- Torresani, L. et al.. (2008). Non-rigid structure-from-motion: Estimating shape and motion with hierarchical priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5), 878–892.
- Toyama, K. (1998). *Prolegomena for Robust Face Tracking*. Technical Report MSR-TR-98-65, Microsoft Research.
- Toyama, K. et al.. (1999). Wallflower: Principles and practice of background maintenance. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 255–261, Kerkyra, Greece.
- Treisman, A. (1985). Preattentive processing in vision. *Computer Vision, Graphics, and Image Processing*, 31(2), 156–177.
- Triggs, B. (1996). Factorization methods for projective structure and motion. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 845–851, San Francisco.
- Triggs, B. (2004). Detecting keypoints with stable position, orientation, and scale under illumination changes. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 100–113, Springer-Verlag, Prague.
- Triggs, B. et al.. (1999). Bundle adjustment — a modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372, Springer, Kerkyra, Greece.

- Trucco, E. and Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*. Prentice Hall.
- Tsai, R. Y. (1987). A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4), 323–344.
- Tschumperlé, D. (2006). Curvature-preserving regularization of multi-valued images using pde's. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 295–307, Springer.
- Tschumperlé, D. and Deriche, R. (2005). Vector-valued image regularization with PDEs: A common framework for different applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27, 506–517.
- Tsin, Y., Kang, S. B., and Szeliski, R. (2003). Stereo matching with reflections and translucency. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2003)*, pages 702–709, Madison, WI.
- Tsin, Y., Ramesh, V., and Kanade, T. (2001). Statistical calibration of CCD imaging process. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 480–487, Vancouver, Canada.
- Tumblin, J., Agrawal, A., and Raskar, R. (2005). Why I want a gradient camera. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 103–110, San Diego, CA.
- Turk, G. and Levoy, M. (1994). Zippered polygonal meshes from range images. *Computer Graphics (SIGGRAPH'94)*, , 311–318.
- Turk, M. and Pentland, A. (1991a). Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1), 71–86.
- Turk, M. and Pentland, A. (1991b). Face recognition using eigenfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 586–591, IEEE Computer Society Press, Maui, Hawaii.
- Tuytelaars, T. and Van Gool, L. (2004). Matching widely separated views based on affine invariant regions. *International Journal of Computer Vision*, 59(1), 61–85.
- Tuytelaars, T. and Mikolajczyk, K. (2007). Local invariant feature detectors. *Foundations and Trends in Computer Graphics and Computer Vision*, 3(1).
- Tuytelaars, T. et al.. (1997). The cascaded Hough transform. In *International Conference on Image Processing (ICIP'97)*, pages 736–739.
- Ullman, S. (1979). The interpretation of structure from motion. *Proceedings of the Royal Society of London, B-203*, 405–426.

- Unnikrishnan, R., Pantofaru, C., and Hebert, M. (2007). Toward objective evaluation of image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), 828–944.
- Unser, M. (1999). Splines: A perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16(6), 22–38.
- Uyttendaele, M., Eden, A., and Szeliski, R. (2001). Eliminating ghosting and exposure artifacts in image mosaics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 509–516, Kauai, Hawaii.
- Uyttendaele, M., Criminisi, A., Kang, S. B., Winder, S., Hartley, R., and Szeliski, R. (2004). Image-based interactive exploration of real-world environments. *IEEE Computer Graphics and Applications*, 24(3), 52–63.
- Vaillant, R. and Faugeras, O. D. (1992). Using extremal boundaries for 3-D object modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 157–173.
- Vaish, V., Szeliski, R., Zitnick, C. L., Kang, S. B., and Levoy, M. (2006). Reconstructing occluded surfaces using synthetic apertures: Shape from focus vs. shape from stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 2331–2338, New York, NY.
- van de Weijer, J. and Schmid, C. (2006). Coloring local feature extraction. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 334–348, Springer.
- van den Hengel, A., Dick, A., Thormhlen, T., Ward, B., and Torr, P. H. S. (2007). Videotrace: Rapid interactive scene modeling from video. *ACM Transactions on Graphics*, 26(3).
- Varma, M. and Ra, D. (2007). Learning the discriminative power-invariance trade-off. In *Tenth International Conference on Computer Vision (ICCV 2007)*, Rio de Janeiro, Brasil.
- Veeraraghavan, A., Raskar, R., Agrawal, A., Mohan, A., and Tumblin, J. (2007). Dappled photography: Mask enhanced cameras for heterodyned light fields and coded aperture refocusing. *ACM Transactions on Graphics*, 26(3).
- Veksler, O. (1999). *Efficient Graph-based Energy Minimization Methods in Computer Vision*. Ph.D. thesis, Cornell University.
- Veksler, O. (2001). Stereo matching by compact windows via minimum ratio cycle. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 540–547, Vancouver, Canada.
- Veksler, O. (2007). Graph cut based optimization for mrfs with truncated convex priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Vergauwen, M. and Van Gool, L. (2006). Web-based 3d reconstruction service. *Machine Vision and Applications*, 17(2), 321–329.

- Vezhnevets, V. *et al.*. (2003). A survey on pixel-based skin color detection techniques. In *GRAPHICON03*, pages 85–92.
- Vicente, S., Kolmogorov, V., and Rother, C. (2008). Graph cut based image segmentation with connectivity priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Vincent, L. and Soille, P. (1991). Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6), 583–596.
- Viola, P. and Wells III, W. (1997). Alignment by maximization of mutual information. *International Journal of Computer Vision*, 24(2), 137–154.
- Viola, P. A. and Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57(2), 137–154.
- Waite, P. and Ferrie, F. (1991). From uncertainty to visual exploration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10), 1038–1049.
- Wallace, G. K. (1991). The JPEG still picture compression standard. *Communications of the ACM*, 34(4), 30–44.
- Wallace, J. R., Cohen, M. F., and Greenberg, D. P. (1987). A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, pages 311–320.
- Waltz, D. L. (1975). Understanding line drawings of scenes with shadows. In Winston, P. H., editor, *The Psychology of Computer Vision*, McGraw-Hill, New York.
- Wang, H. and Oliensis, J. (2008). Shape matching by segmentation averaging. In *Tenth European Conference on Computer Vision (ECCV 2008)*, pages 562–575, Springer-Verlag, Marseilles.
- Wang, J. and Cohen, M. F. (2005a). An iterative optimization approach for unified image segmentation and matting. In *Tenth International Conference on Computer Vision (ICCV 2005)*, Beijing, China.
- Wang, J. and Cohen, M. F. (2005b). An iterative optimization approach for unified image segmentation and matting. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 936–943, Beijing, China.
- Wang, J. and Cohen, M. F. (2007a). Image and video matting: A survey. *Foundations and Trends in Computer Graphics and Computer Vision*, 3(2).
- Wang, J. and Cohen, M. F. (2007b). Optimized color sampling for robust matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.

- Wang, J. and Cohen, M. F. (2007c). Simultaneous matting and compositing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Wang, J., Agrawala, M., and Cohen, M. F. (2007). Soft scissors: An interactive tool for realtime high quality matting. *ACM Transactions on Graphics*, 26(3).
- Wang, J., Thiesson, B., Xu, Y., and Cohen, M. (2004). Image and video segmentation by anisotropic kernel mean shift. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 238–249, Springer-Verlag, Prague.
- Wang, J., Bhat, P., Colburn, R. A., Agrawala, M., and Cohen, M. F. (2005). Video cutout. *ACM Transactions on Graphics*, 24(3), 585–594.
- Wang, J. Y. A. and Adelson, E. H. (1994). Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5), 625–638.
- Wang, L., Kang, S. B., Szeliski, R., and Shum, H.-Y. (2001). Optimal texture map reconstruction from multiple views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 347–354, Kauai, Hawaii.
- Wang, Y. and Zhu, S.-C. (2003). Modeling textured motion: Particle, wave and sketch. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 213–220, Nice, France.
- Wang, Z., Bovik, A. C., and Simoncelli, E. P. (2005). Structural approaches to image quality assessment. In Bovik, A. C., editor, *Handbook of Image and Video Processing*, pages 961–974, Elsevier Academic Press.
- Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 600–612.
- Ward, G. (1992). Measuring and modeling anisotropic reflection. *Computer Graphics (Proceedings SIGGRAPH'92)*, 26(4), 265–272.
- Ward, G. (1994). The radiance lighting simulation and rendering system. *Computer Graphics (SIGGRAPH'94)*, , 459–472.
- Ward, G. (2003). Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *Journal of Graphics Tools*, 8(2), 17–30.
- Ward, G. (2004). High dynamic range image encodings. http://www.anyhere.com/gward/hdrenc/hdr_encodings.html.
- Watt, A. (1995). *3D Computer Graphics*. Addison-Wesley, third edition.
- Weber, J. and Malik, J. (1995). Robust computation of optical flow in a multi-scale differential framework. *International Journal of Computer Vision*, 14(1), 67–81.

- Weber, M., Welling, M., and Perona, P. (2000). Unsupervised learning of models for recognition. In *Sixth European Conference on Computer Vision (ECCV 2000)*, pages 18–32, Springer-Verlag, Dublin, Ireland.
- Wei, C. Y. and Quan, L. (2004). Region-based progressive stereo matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 106–113, Washington, D. C.
- Wei, L.-Y. and Levoy, M. (2000). Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000*, , 479–488. ISBN 1-58113-208-5.
- Weickert, J. (1998). *Anisotropic Diffusion in Image Processing*. Tuebner, Stuttgart.
- Weickert, J. et al.. (1998). Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Transactions on Image Processing*, 7(3), 398–410.
- Weiss, Y. (1999). Segmentation using eigenvectors: A unifying view. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 975–982, Kerkyra, Greece.
- Weiss, Y. (2001). Deriving intrinsic images from image sequences. In *Eighth International Conference on Computer Vision (ICCV 2001)*, pages 7–14, Vancouver, Canada.
- Weiss, Y. and Adelson, E. H. (1996). A unified mixture framework for motion segmentation: Incorporating spatial coherence and estimating the number of models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 321–326, San Francisco.
- Weiss, Y. and Freeman, B. (2007). What makes a good model of natural images? In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Welch, G. and Bishop, G. (1995). *An Introduction to the Kalman Filter*. Technical Report TR 95-041, Department of Computer Science, University of North Carolina at Chapel Hill.
- Wells, III, W. M. (1986). Efficient synthesis of gaussian filters by cascaded uniform filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2), 234–239.
- Weng, J., Ahuja, N., and Huang, T. S. (1993). Optimal motion and structure estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9), 864–884.
- Werner, T. and Zisserman, A. (2002). New techniques for automated architectural reconstruction from photographs. In *Ninth European Conference on Computer Vision (ECCV 2006)*, pages 541–555, Springer-Verlag, Copenhagen.
- Westin, S. H. et al.. (1992). Predicting reflectance functions from complex surfaces. *Computer Graphics (Proceedings SIGGRAPH'92)*, 26(4), 255–264.

- Wexler, Y., Fitzgibbon, A., and Zisserman, A. (2002). Bayesian estimation of layers from multiple images. In *Seventh European Conference on Computer Vision (ECCV 2002)*, pages 487–501, Springer-Verlag, Copenhagen.
- Williams, D. and Burns, P. D. (2001). Diagnostics for digital capture using MTF. In *IS&T PICS Conference*, pages 227–232, Society for Imaging Science and Technology.
- Williams, L. (1978). Pyramidal parametrics. *Computer Graphics*, 12(3), 270–274.
- Williams, L. (1983). Pyramidal parametrics. *Computer Graphics*, 17(3), 1–11.
- Williams, L. (1990). Performance driven facial animation. *Computer Graphics*, 24(4), 235–242.
- Williams, O., Blake, A., and Cipolla, R. (2003). A sparse probabilistic learning algorithm for real-time tracking. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 353–360, Nice, France.
- Williams, T. L. (1999). *The Optical Transfer Function of Imaging Systems*. Institute of Physics Publishing, London.
- Winder, S. and Brown, M. (2007). Learning local image descriptors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Winkenbach, G. and Salesin, D. H. (1994). Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94*, pages 91–100, ACM SIGGRAPH / ACM Press, Orlando, Florida. ISBN 0-89791-667-0.
- Winn, J. and Shotton, J. (2006). The layout consistent random field for recognizing and segmenting partially occluded objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 37–44, New York City, NY.
- Winston, P. H., editor. (1975). *The Psychology of Computer Vision*, McGraw-Hill, New York.
- Witkin, A., Terzopoulos, D., and Kass, M. (1986). Signal matching through scale space. In *Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 714–719, Morgan Kaufmann Publishers, Philadelphia.
- Witkin, A., Terzopoulos, D., and Kass, M. (1987). Signal matching through scale space. *International Journal of Computer Vision*, 1, 133–144.
- Witkin, A. P. (1981). Recovering surface shape and orientation from texture. *Artificial Intelligence*, 17, 17–45.
- Witkin, A. P. (1983). Scale-space filtering. In *Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 1019–1022, Morgan Kaufmann Publishers.

- Wolberg, G. (1990). *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos.
- Wolberg, G. and Pavlidis, T. (1985). Restoration of binary images using stochastic relaxation with annealing. *Pattern Recognition Letters*, 3, 375–388.
- Wolff, L. B., Shafer, S. A., and Healey, G. E., editors. (1992a). *Radiometry. Physics-Based Vision: Principles and Practice*, Jones & Bartlett, Cambridge, MA.
- Wolff, L. B., Shafer, S. A., and Healey, G. E., editors. (1992b). *Shape Recovery. Physics-Based Vision: Principles and Practice*, Jones & Bartlett, Cambridge, MA.
- Wood, D. N. et al.. (1997). Multiperspective panoramas for cel animation. In *Computer Graphics Proceedings, Annual Conference Series*, pages 243–250, ACM SIGGRAPH, Proc. SIGGRAPH'97 (Los Angeles).
- Wood, D. N., Azuma, D. I., Aldinger, K., Curless, B., Duchamp, T., Salesin, D. H., and Stuetzle, W. (2000). Surface light fields for 3d photography. *Proceedings of SIGGRAPH 2000*, , 287–296.
- Woodford, O., Reid, I., Torr, P. H., and Fitzgibbon, A. (2008). Global stereo reconstruction under second order smoothness priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Woodham, R. J. (1981). Analysing images of curved surfaces. *Artificial Intelligence*, 17, 117–140.
- Wright, S. (2006). *Digital Compositing for Film and Video*. Focal Press, 2nd edition.
- Wyszecki, G. and Stiles, W. S. (1982). *Color Science: Concepts and Methods, Quantitative Data and Formulae*. John Wiley & Sons, New York, 2nd edition.
- Wyszecki, G. and Stiles, W. S. (2000). *Color Science: Concepts and Methods, Quantitative Data and Formulae*. John Wiley & Sons, New York, 2nd edition.
- Xiao, J. and Shah, M. (2003). Two-frame wide baseline matching. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 603–609, Nice, France.
- Xiong, Y. and Turkowski, K. (1997). Creating image-based VR using a self-calibrating fisheye lens. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 237–243, San Juan, Puerto Rico.
- Xiong, Y. and Turkowski, K. (1998). Registration, calibration and blending in creating high quality panoramas. In *IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 69–74, IEEE Computer Society, Princeton.
- Yang, D. et al.. (1999). A 640x512 CMOS image sensor with ultra-wide dynamic range floating-point pixel level ADC. *IEEE Journal of Solid State Circuits*, 34(12), 1821–1834.

- Yang, L., Jin, R., Sukthankar, R., and Jurie, F. (2008). Unifying discriminative visual codebook generation with classifier training for object category recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Yang, Y., Yuille, A., and Lu, J. (1993). Local, global, and multilevel stereo matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 274–279, IEEE Computer Society, New York.
- Yaou, M.-H. and Chang, W.-T. (1994). Fast surface interpolation using multiresolution wavelets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7), 673–689.
- Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2000). Generalized belief propagation. In *Advances in Neural Information Processing Systems 13*, pages 689–695, MIT Press.
- Yezzi, Jr., A. J., Kichenassamy, S., Kumar, A., Olver, P., and Tannenbaum, A. (1997). A geometric snake model for segmentation of medical imagery. *IEEE Transactions on Medical Imaging*, 16(2), 199–209.
- Yin, P., Criminisi, A., Winn, J., , and Essa, I. (2007). Tree-based classifiers for bilayer video segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, Minneapolis, MN.
- Yoon, K.-J. and Kweon, I.-S. (2005). Locally adaptive support-weight approach for visual correspondence search. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, pages 924–931, San Diego, CA.
- Yoon, K.-J. and Kweon, I. S. (2006). Stereo matching with symmetric cost functions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 2371–2377, New York City, NY.
- Yu, S. X. and Shi, J. (2003). Multiclass spectral clustering. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 313–319, Nice, France.
- Yu, Y. and Malik, J. (1998). Recovering photometric properties of architectural scenes from photographs. *Computer Graphics (SIGGRAPH'96)*, , 207–218.
- Yu, Y.,Debevec, P., Malik, J., and Hawkins, T. (1999). Inverse global illumination: Recovering reflectance models of real scenes from photographs. *Proceedings of SIGGRAPH 99*, , 215–224. ISBN 0-20148-560-5. Held in Los Angeles, California.
- Yuan, L., Sun, J., Quan, L., and Shum, H.-Y. (2007). Image deblurring with blurred/noisy image pairs. *ACM Transactions on Graphics*, 26(3).
- Yuan, L., Sun, J., Quan, L., and Shum, H.-Y. (2008). Progressive inter-scale and intra-scale non-blind image deconvolution. *ACM Transactions on Graphics*, 27(3).

- Yuille, A. L. and Poggio, T. (1984). *A Generalized Ordering Constraint for Stereo Correspondence*. A. I. Memo 777, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.
- Zabih, R. and Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In *Third European Conference on Computer Vision (ECCV'94)*, pages 151–158, Springer-Verlag, Stockholm, Sweden.
- Zelnik-Manor, L. and Perona, P. (2007). Automating joiners. In *Symposium on Non Photorealistic Animation and Rendering*, ACM SIGGRAPH, Annecy.
- Zhang, G., Jia, J., Wong, T.-T., and Bao, H. (2008). Recovering consistent video depth maps via bundle optimization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Zhang, J. *et al.*. (2007). Local features and kernels for classification of texture and object categories: a comprehensive study. *International Journal of Computer Vision*, 73(2), 213–238.
- Zhang, L. *et al.*. (2002). Single view modeling of free-form scenes. *Journal of Visualization and Computer Animation*, 13(4), 225–235.
- Zhang, L., Curless, B., and Seitz, S. (2003). Spacetime stereo: Shape recovery for dynamic scenes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2003)*, pages 367–374, Madison, WI.
- Zhang, L., Snavely, N., Curless, B., and Seitz, S. M. (2004). Spacetime faces: high resolution capture for modeling and animation. *ACM Transactions on Graphics*, 23(3), 548–558.
- Zhang, R., Tsai, P.-S., Cryer, J., and Shah, M. (1999). Shape-from-shading: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(1), 690–706.
- Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2).
- Zhang, Z. (1998a). Determining the epipolar geometry and its uncertainty: A review. *International Journal of Computer Vision*, 27(2), 161–195.
- Zhang, Z. (1998b). On the optimization criteria used in two-view motion analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(7), 717–729.
- Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientations. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 666–687, Kerkyra, Greece.
- Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 1330–1334.

- Zhang, Z. and He, L.-W. (2007). Whiteboard scanning and image enhancement. *Digital Signal Processing*, 17(2), 414–432.
- Zhang, Z. *et al.* (1995). A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78, 87–119.
- Zhang, Z. and Shan, Y. (2000). A progressive scheme for stereo matching. In Pollefeys, M. *et al.*, editors, *Second European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE 2000)*, pages 68–85, Dublin, Ireland.
- Zheng, J. Y. (1994). Acquiring 3-D models from sequences of contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2), 163–178.
- Zheng, Y., Lin, S., and Kang, S. B. (2006). Single-image vignetting correction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2006)*, pages 461–468, New York City, NY.
- Zheng, Y., Yu, J., Kang, S.-B., Lin, S., and Kambhamettu, C. (2008). Single-image vignetting correction using radial gradient symmetry. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, AK.
- Zheng, Y., Zhou, X. S., Georgescu, B., Zhou, S. K., and Comaniciu, D. (2006). Example based non-rigid shape detection. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 423–436, Springer.
- Zhong, J. and Sclaroff, S. (2003). Segmenting foreground objects from a dynamic, textured background via a robust kalman filter. In *Ninth International Conference on Computer Vision (ICCV 2003)*, pages 44–50, Nice, France.
- Zhu, S.-C. and Mumford, D. (2006). A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Computer Vision*, 2(4).
- Zhu, S. C. and Yuille, A. L. (1996). Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9), 884–900.
- Zitnick, C. L. and Kanade, T. (2000). A cooperative algorithm for stereo matching and occlusion detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7), 675–684.
- Zitnick, C. L., Jojic, N., and Kang, S. B. (2005). Consistent segmentation for optical flow estimation. In *Tenth International Conference on Computer Vision (ICCV 2005)*, pages 1308–1315, Beijing, China.
- Zitnick, C. L., Kang, S. B., Uyttendaele, M., Winder, S., and Szeliski, R. (2004). High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics*, 23(3), 600–608.

Zitov'aa, B. and Flusser, J. (2003). Image registration methods: A survey. *Image and Vision Computing*, 21, 997–1000.

Zoghalmi, I., Faugeras, O., and Deriche, R. (1997). Using geometric corners to build a 2D mosaic from a set of images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 420–425, San Juan, Puerto Rico.

Zomet, A. and Peleg, S. (2002). Multi-sensor super resolution. In *IEEE Workshop on Applications of Computer Vision*, pages 27–31, IEEE Computer Society, Orlando.

Zongker, D. E., Werner, D. M., Curless, B., and Salesin, D. H. (1999). Environment matting and compositing. *Proceedings of SIGGRAPH 99*, , 205–214. ISBN 0-20148-560-5. Held in Los Angeles, California.

[Note: Bibliography appears to be sorted by full author names, so that initials matter. Is there a way to sort it by the way the citation appears in the text? Look in bookstyl.bst]

[Note: Months are missing. Is it possible to have these while maintaining the author/year format? Check IJCV.]

[Note: Drop publisher from conference proceedings?]

[Note: Add a separate section at start of bibliography on most important journals and conferences, both as a reference (although this could go in intro), and to enable abbreviations? Would then have to re-do .bib files to symbolically reference conferences and journals.]

[Note: Try to replace all conference and TR versions with journal versions. The USC CV bibliography and CiteSeer should help. It would also be great to sanity check all of the references again the USC/Price bibliography, but may be easiest to do once my .bib files are ingested into some kind of a database.]

[Note: This is just the index for the Intro and Image Formation sections for now, just to see how indexing works...]

Index

- Active appearance models, 268
- Affine transforms, 37
- Algorithms
 - testing, xvi
- Aliasing, 77
- Ambient illumination, 65
- Analog to digital conversion (ADC), 77
- Anti-aliasing filter, 78
- Aperture, 69
- Applications, 5
 - frame interpolation, 360
 - morphing, 359
 - mosaic-based video compression, 377
 - motion-based user interface, 362
 - panography, 312
 - planar pattern tracking, 318
 - video stabilization, 356
- VideoMouse, 318
- whiteboard and document scanning, 372
- Aspect ratio, 52
- Automatic gain control (AGC), 75
- Axis/angle representation of rotations, 41
- B-snakes, 268
- Barrel distortion, 59
- Bayer pattern (RGB sensor mosaic), 85
 - de-mosaicing, 86, 438
- Blur kernel, 69
 - estimation, 416
- Body color, 63
- BRDF, 62
- Brightness constancy, 3
- Calibration matrix, 51
- Camera
 - calibration, 49, 96
 - intrinsic, 49
 - matrix, 54
- Catadioptric optics, 71
- Chromatic aberration, 70
- Chromaticity coordinates, 83
- CIE L*a*b*, *see* color
- CIE XYZ, *see* color
- Circle of confusion, 69
- Co-vectors, 37
- Collineation, 40
- Color, 80
 - balance, 86, 95
 - cameras, 84
 - de-mosaicing, 86, 438
 - HSV, 91
 - L*a*b*, 84
 - primaries, 81
 - ratios, 91
 - RGB, 81
 - twist, 86
 - XYZ, 81
 - YIQ, 88
 - YUV, 88
- Color filter array (CFA), 85
- Compression, 91
- Conic sections, 33

- connected components, 129
- De-mosaicing (Bayer), 86, 438
- Depth of field, 69, 94
- Diffuse reflection, 63
- Digital cameras, 73
 - color, 84
 - color filter arrays (CFA), 85
 - compression, 91
- Discrete cosine transform (DCT), 92, 143
- Elastic nets, 267
- Environment map, 61
- Euler angles, 41
- Exponential twist, 43
- Exposure value (EV), 70
- F-number (stop), 69, 94
- Fill factor, 75
- Fisheye lens, 59
- Flash matting, 450
- Focal length, 68
- Gamma correction, 87, 95
- Gaussian mixtures, *see* Mixture of Gaussians
- Generalized cylinders, 11
- Geometric primitives
 - points, lines, and planes, 32
- Geometric transformations
 - 2D, 35, 93
 - 3D, 39, 94
 - affine, 37
 - calibration matrix, 51
 - collineation, 40
 - hierarchy, 37
 - perspective, 37
 - projections, 46
 - projective, 37
 - rotations, 40
- similarity, 36
- translation, 35
- Global illumination, 66
- History of computer vision, 11
- Homography, 37
- Ideal points, 32
- Illusions, 3
- Image compression, 91
- Image formation
 - geometric, 32
 - photometric, 60
- Image segmentation, *see* segmentation
 - active contours, 266
 - binary MRF, 296
 - graph cuts, 296
 - level sets, 276
- image segmentation
 - binary MRF, 184
 - connected components, 129
 - thresholding, 126
- Intrinsic images, 12
- Inverse problems, 3
- ISO setting, 75
- K-means, 286
- L*a*b*, *see* color
- Lambertian reflection, 63
- Least squares, 93
- Lens
 - compound, 70
 - thin, 68
- Lens distortions, 58
- Lens law, 68
- Level sets, 276
- Lightness, 84
- Line at infinity, 32

- Line labeling, 11
- Luminance, 82
- Mahalanobis distance, 286, 522
- Marr's framework, 12
 - computational theory, 13
 - hardware implementation, 13
 - representations and algorithms, 13
- Matte reflection, 63
- Matting
 - flash, 450
 - shadow, 450
 - smoke, 450
- Metamers, 82
- Mixture of Gaussians, 286
- Models
 - forward, 3
 - physics-based, 3
 - probabilistic, 3
- Modulation transfer function (MTF), 80
- Morphing, 359
- Mosaics
 - video compression, 377
 - whiteboard and document scanning, 372
- Motion compensation, 92
- Motion estimation
 - user interface applications, 362
- Nodal point, 70
- Noise
 - sensor, 75
- Noise level function, 76, 95
- Normal vector, 34
- Normalized device coordinates (NDC), 49
- Nyquist rate / frequency, 78
- Object-centered projection, 57
- Optical center, 52
- Optical illusions, 3
- Optical transfer function (OTF), 80
- Optics, 68
 - chromatic aberration, 70
 - vignetting, 71
- Orthographic projection, 46
- Paraperspective projection, 48
- perspective projection, 48
- Perspective transform (2D), 37
- Phong shading, 65
- Photometric image formation, 60
 - global illumination, 66
 - lighting, 60
 - optics, 68
 - radiosity, 66
 - reflectance, 61
 - shading, 65
- Photometry, 60
- Pincushion distortions, 59
- Plücker coordinates, 35
- Planar pattern tracking, 318
- Plane at infinity, 34
- Plane plus parallax, 54
- Point spread function, 78
 - estimation, 416
- Points at infinity, 32
- Polar coordinates, 33
- Pop-out effect, 4
- Projections
 - object-centered, 57
 - orthographic, 46
 - para-perspective, 48
- projections
 - perspective, 48
- Quaternions, 43

- Radial distortion, 58
Radiometric image formation, 60
Radiometry, 60
Radiosity, 66
Ray tracing, 66
Rectangle detection, 263
Reflectance, 61
Reflection
 diffuse, 63
 specular, 64
Region segmentation, *see* segmentation
RGB (red green blue), *see* color
Rodriguez's formula, 42
Rotations, 40
 Euler angles, 41
 axis/angle, 41
 exponential twist, 43
 incremental, 45
 interpolation, 45
 quaternions, 43
 Rodriguez's formula, 42
Sampling, 77
Segmentation, 278
 k-means, 286
 Mixture of Gaussians, 286
Sensing, 73
 aliasing, 77
 color, 80
 color balance, 86
 gamma, 87
 sampling, 77
 sampling pitch, 74
Sensor noise, 75
Shading, 65
 equation, 64
Shadow matting, 450
Shutter speed, 74
Similarity transform, 36
Skin detection, 96
Smoke matting, 450
Snake shape priors, 268
Spectral response function, 85
Spectral sensitivity, 85
Specular reflection, 64
Spherical coordinates, 34
Spherical linear interpolation, 45
Stitching
 cylindrical, 378
 panography, 312
Testing algorithms, xvi
Thin lens, 68
thresholding, 126
Tri-chromatic sensing, 81
Video de-noising, 500
Video stabilization, 356
Videomouse, 318
Vignetting, 71
Visual illusions, 3
White balance, 86, 95
XYZ, *see* color