Synthesis, Editing, and Rendering of Multiscale Textures

Charles Han

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2011

ABSTRACT

Synthesis, Editing, and Rendering of Multiscale Textures

Charles Han

The study of textures—images with repeated visual content—has produced a number of useful tools and algorithms for analysis, synthesis, editing, rendering, and a variety of other applications. However, the recent rapid growth in data storage and computational abilities has expanded the notion of what constitutes a texture. Modern textures can often outstrip traditional assumptions on input size by several orders of magnitude. Additionally, these *multiscale* textures typically contain features at not just one scale but rather across a wide range of scales, further violating existing assumptions.

In order to meaningfully capture the large-scale features present in multiscale textures, we introduce a new example-based input representation, the *exemplar graph*. This representation enables allows us to efficiently define textures spanning a large–or possibly infinite–range of visual scales. We develop a hierarchical, parallelizable algorithm for performing texture synthesis from an input exemplar graph.

In addition to automated generation, an increasingly important application of texture synthesis is in interactive tools for guiding texture design. This modality is especially important for multiscale textures, as they offer special perceptual challenges to artists. We examine algorithmic and engineering optimizations to enable real-time analysis and synthesis of multiscale textures, and explore potential implications for editing tools.

Finally, we study the issue of display. To accurately view a large image at distance, some filtering operation must be performed. In many cases, such as traditional color images, the filtering operations are well-known. However, other texture representations, such as normal or displacement maps, present special difficulties for filtering. We treat the former case, presenting a principled analysis and algorithms for filtering and display of large normal maps.

TABLE OF CONTENTS

LIST OF FIGURES iv

LIST OF ALGORITHMS vi

ACKNOWLEDGMENTS vii

NOTATION ix

I BACKGROUND

1 Introduction 2

2 Example-driven Texture Models 6

2.1 Markov Random Field 6

2.2 Statistical Models 8

2.3 Other Models 9

3 Approach and Overview 11

II AUTHORING MULTISCALE TEXTURES

4 The Exemplar Graph 14

4.1 Definition 15

4.2 Inconsistency 16

5 Synthesis 19

5.1 Related Work 20

5.2 Multiscale Texture Synthesis 21

5.2.1 Data structures 21

5.2.2 Algorithm 23

5.3 Inconsistency Correction 25

5.4 GPU optimization . . . 27
5.5 Results . . . 29
6 Editing 35
6.1 Related Work . . . 36
6.2 System overview . . . 38
6.3 Incremental PCA . . . 39
6.3.1 Derivation . . . 40
6.3.2 Computation . . . 41
6.3.3 Sparse neighborhood sampling . . . 44
6.3.4 PCA and neighborhood projection . . . 44
6.4 Incremental PatchMatch . . . 45
6.4.1 Notation and Background . . . 45
6.4.2 Restricted passes . . . 47
6.4.3 Biased search . . . 47
6.4.4 Vertical propagation . . . 48
6.5 Implementation and Results . . . 49
III FILTERING OF MULTISCALE TEXTURES
7 Normal Map Filtering 53
7.1 Related Work . . . 56
7.2 Preliminaries . . . 58
7.2.1 BRDF representation and parameterization . . . 59
7.2.2 Normal map representation and filtering . . . 60
7.3 Normal Mapping as Convolution . . . 60
7.3.1 Normal distribution function . . . 61
7.3.2 Frequency-domain analysis in 2D . . . 61
7.3.3 Frequency-domain analysis in 3D . . . 62
7.4 Spherical Harmonics . . . 65
7.4.1 Algorithm . . . 65
7.4.2 Results . . . 66

7.5 Spherically Symmetric Distributions . . . 68
7.5.1 Basic theoretical framework for using SRBFs . . . 69
7.5.2 Discussion: unifying framework and multiscale . . . 69
7.5.3 Choice of radial basis function . . . 70
7.6 Von Mises-Fisher Mixtures . . . 71
7.6.1 Fitting NDFs with mixtures of vMFs . . . 72
7.6.2 Spherical harmonic coefficients for rendering . . . 74
7.6.3 Complex lighting . . . 76
7.6.4 Extensions . . . 77
7.6.5 Results . . . 81

IV CONCLUSIONS

8 Future Directions 85

9 Summary and Final Words 87

BIBLIOGRAPHY 89

LIST OF FIGURES

1.1 Multiscale texture 3
1.2 The mid-frequency problem 4

4.1 The exemplar graph 15
4.2 A simple graph 16
4.3 Inconsistency correction 17

5.1 Data structures 21
5.2 Transfer functions 26
5.3 Coherent infinite zooms 30
5.4 Super-resolution 31
5.5 Compact representation 32
5.6 A simple chain 33

6.1 Editing system overview 38
6.2 PatchMatch phases 46
6.3 Algorithm comparison 50
6.4 Texture editing sessions 51

7.1 The normal map filtering problem 54
7.2 Comparison of filtering methods 55
7.3 Spherical harmonic anisotropic filtering 64
7.4 Temporal coherence 67
7.5 Rendering with acquired BRDFs 68

7.6 Multiscale tradeoffs 70
7.7 vMF lobe fitting 73
7.8 Dynamically changing reflectance 82
7.9 Normal map filtering under complex lighting 83

LIST OF ALGORITHMS

6.1 Space-optimized covariance computation 43
7.1 The Spherical EM algorithm 74
7.2 Pseudocode for the vMF GLSL fragment shader 80

ACKNOWLEDGMENTS

This thesis would have been impossible without: my co-advisors Eitan Grinspun[1] and Ravi Ramamoorthi[2]; mentor Hugues Hoppe[3]; committee members Peter Belhumeur and Shree Nayar[4]; fellow Columbia students (in order of appearance) Aner Ben-Artzi, Bo Sun, Ryan Overbeck, David Harmon, Miklós Bergou, Kevin Egan, Etienne Vouga, and Breannan Smith[5]; collaborator Eric Risser[6]; the fine artists, researchers, and developers at Weta Digital[7]; admins Anne Fleming, Lily Secora, and Jessica Rosa[8]; Daisy Nguyen and CRF[9]; Bob, Fang, Greg, Jenn, Jess, Job, Kaliq, Max, Sherry, Steph, and the rest of my urban family[10]; long-distance pals Caroline, Dave, Wendy, and so many more[11]; Girlfriend[12]; my little sister Naly[13] and her little man Milo[14]; and my parents[15].

Thank you all for carrying me through this journey.

---

[1] An ever-positive wellspring of guidance through my many distractions, diversions, and disillusionments. Wow, I sure didn't make it easy for you, did I? We wound up a long way from where we started, and I wonder if I would've made it with any other primary-care advisor on Earth. Thanks for the thrilling ride.

[2] Always ready with the perfect bit of insight or motivation, you never let us drift too far off into the clouds. Conducting research with you has been at once humbling and empowering.

[3] Ours has been a truly inspiring collaboration. It's been a joy to witness firsthand your unique blend of brilliance, intuition, and practicality; I hope to someday emulate it in my own work.

[4] This thesis owes much to your incisive questions and urgings to dig deeper. Thank you for your direction and perspective.

[5] I am proud to call you my contemporaries, and more importantly my friends. Thanks for the many thoughtful discussions, spirited collaborations, and crazy SIGGRAPH memories (or lack thereof).

[6] A sharp research mind and all around nice guy. Also, I still owe you big-time for that rescue in Dublin!

[7] So many of the ideas in this research—and so many more yet to be properly explored—arose from my time in Wellington. I especially thank Joe Letteri for inviting me, and Peter Hillman and Richard Addison-Wood for countless thought-provoking conversations.

[8] It's scary to think how many reimbursements and registration deadlines I would have missed if not for you. Thank you each so much for your eternal patience.

[9] True heroes, putting out fires every day (literally, when the chill water goes down).

[10] I'm sure I missed a few names, sorry! Whether it was throwing a house party at The ☞, going [food] clubbing, enjoying a weekly Wednesday drink, or just plain hanging out, my time in New York has been a constant blast. A big, ZONG thank you to the best friends in world!

[11] . . . the *other* best friends in the world!

[12] Susanna Gyujin Kim, I can't thank you enough for being my constant companion / muse / fashion model / cheerleader / study buddy / coach / Thai masseuse / friend.

[13] Really the grown-up one between us, whom I look up to in many ways. Expect me to come asking for advice one day when I have a kid or a tax problem.

[14] Listen to your mother, she loves you very much!

[15] Thank you for giving so completely of yourselves so that I could have every opportunity. I am all I am because of you.

NOTATION

ACRONYM	DEFINITION	SYMBOL
MRF	Markov random field	
	exemplar	E^i
	Gaussian stack level	E^i_k
	admissible candidates	$\mathcal{A}(E^i_k)$
	appearance transfer function	$\boldsymbol{r}$
	cumulative transfer function	$\boldsymbol{R}$
NNF	nearest neighbor field	f
NDF	normal distribution function	$\gamma(\boldsymbol{n})$
SH	spherical harmonics	Y_{lm}
SRBF	spherical radial basis function	$\gamma(\boldsymbol{n} \cdot \boldsymbol{\mu})$
EM	expectation maximization	
vMF	von Mises-Fisher distribution	$\gamma(\boldsymbol{n} \cdot \boldsymbol{\mu}; \theta)$, $\theta = \{\kappa, \mu\}$
movMF	mixture of vMF lobes	$\gamma(\boldsymbol{n}; \Theta)$, $\Theta = \{\alpha_j, \theta_j\}_{j=1}^{J}$

To Kyung-Sook and Dong Suk Han
for their unwavering love and support

Part I

BACKGROUND

CHAPTER 1

INTRODUCTION

Texture is an essential part of our everyday experience. From the structured mortar lines of a brick wall, to the seemingly haphazard strands of a shaggy carpet, texture gives us important cues in identifying and understanding the world around us. It is no surprise, then, that the visual representation of texture has long been an important focus both in computer graphics research and in industry.

Indeed, the demand for textures is greater now than ever. As graphics tools and practices have improved, there has been increasing trust and reliance on digital effects in industry. It is no longer uncommon for movies and games to require the creation (and texturing) of entire digitally created characters, sets, and worlds. Alongside this growing need for *more* textures, there is simultaneously a need for *better* textures. Display capabilities and data availability are constantly climbing, driving up resolution demands. Furthermore, these larger texture sizes lead naturally to greater visual complexity. In short, we find ourselves ever in need of more textures, larger textures, and higher-quality textures.

This last requirement—*higher-quality*—is particularly challenging. Consider for instance the geographic texture in Figure 4.1. When viewed from satellite distance, the texture features are on the order of oceans and land masses. As we zoom in (bottom insets), features take on the shape of coastlines, forests, or mountain ranges. At the finest levels (top insets) we begin to differentiate rivers, valley systems, and individual ridges.

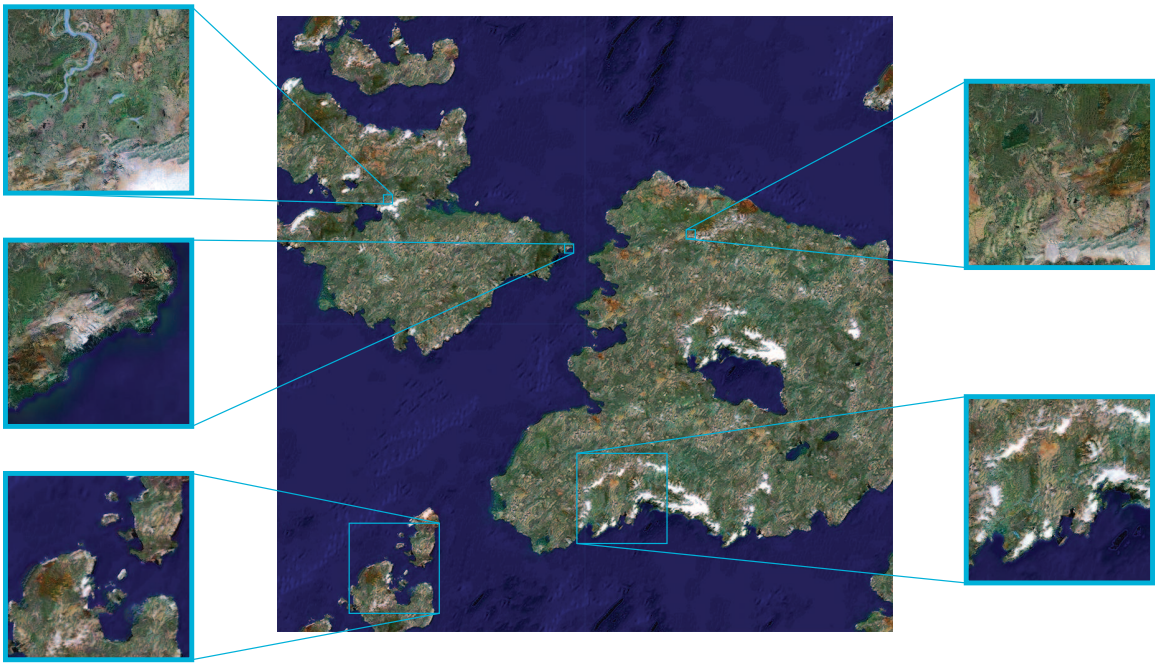


Figure 1.1: **Multiscale texture.** Characteristic of many modern textures, this $16k \times 16k$ texture exhibits features at a broad range of scales.

This example illustrates a *multiscale texture*–that is, a texture that simultaneously contains features of vastly varying size. While it is difficult to precisely quantify what makes a texture multiscale[1], it is easy to see that such a beast reveals several shortcomings in the way we currently deal with textures.

Challenges for tools Multiscale textures present special technical difficulties, and therefore we cannot simply import our existing tools. As we will cover in greater detail in the next section, most all texture models incorporate some fixed notion of feature size. This limits the range of scales that can be represented, and thus clearly presents a problem in the multiscale setting. Revisiting our Figure 4.1 texture, note that an "appropriate" setting of feature size would need to be several thousands of pixels wide—several orders of magnitude larger than the usual notions. Even supposing that computational demands were no issue, such an approach would still be conceptually wasteful. In representing

[1] how tall is the world's shortest giant?

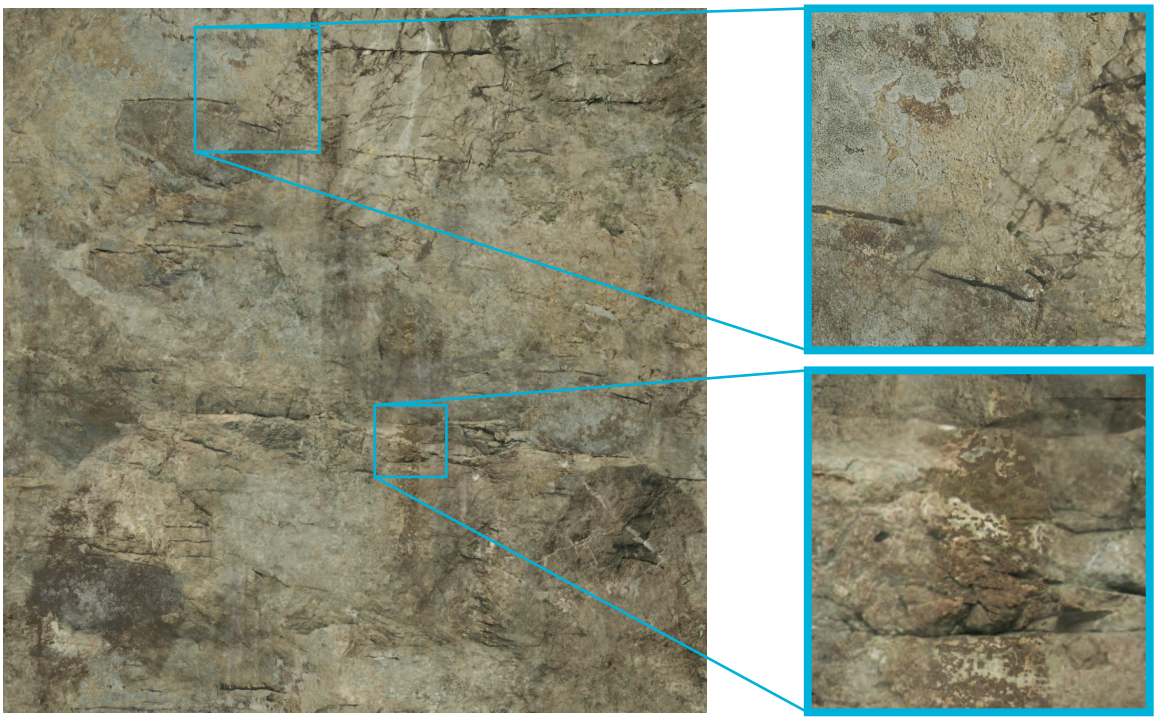


Figure 1.2: **The mid-frequency problem.** Manual texture authoring presents a perceptual hurdle for human artists. Although coarse layouts and fine details are usually well-handled, resolving all frequencies *simultaneously* proves to be difficult.

the largest textural features, a great deal of effort would (for most real-world textures) be wasted on redundant information at finer scales.

Dissatisfaction with these limitations has led to recent research interest in *inhomogeneous* textures [Zhang *et al.*, 2003; Wei *et al.*, 2008; Rosenberger *et al.*, 2009]. These methods augment the traditional models of texture to accommodate a variable coarse structure, driven by the insight that most modern useful textures contain interesting variation at both fine and coarse scales. Our research in multiscale textures follows naturally from a generalized form of this view: texturing tools should accommodate meaningful features at a *broad range* of scales.

Challenges for artists Faced with the aforementioned technical hurdles, current industry practice has turned to a time-honored solution: manual labor. Often using traditional computer painting tools, human artists are able to perform a rough sort of "texture

synthesis". Much to their credit, the results can generally be of high subjective quality, but obvious problems remain with this strategy. First, this is often a tedious and labor-intensive (read: costly) process. By its very nature, texture editing is a repetitive chore; we would rather allow the artist to concentrate on broad aesthetic decisions rather than narrow technical ones.

More fundamentally, there exists a phenomenon that we informally deem the *mid-frequency problem*: people are adept at editing coarse frequencies (*e.g.*, laying out broad patches of texture) and fine frequencies (pixel-level edits using a Photoshop-type tool), but often have difficulties managing the frequencies in between. This will manifest as noticeable irregularities, as we show in Figure 1.2. This is a real texture asset taken from a motion picture pipeline; we see that although the artist has assembled a convincing coarse appearance and has preserved many fine details, numerous "patching" artifacts remain in the final result. This problem is both technical and perceptual. A painting tool such as Photoshop offers excellent low-level control but does not generally give any facility to manage characteristic structures—precisely the task at which texture synthesis excels. More crucially, people tend to think naturally in terms of a small band of frequencies at a given time [Julesz, 1981], making it counterintuitive for an artist to simultaneously work at all scales when painting.

Given that our texture needs are rapidly outgrowing both existing tools and human capabilities, it becomes imperative that we develop new "multiscale-ready" methods; this thesis aims to be a first step in this direction. We organize our document as follows. In the remainder of this chapter we will provide a survey of relevant models for texture. In Chapter 5 we introduce new data structures and algorithms for the efficient specification and synthesis of multiscale textures. We further build on these concepts in Chapter 6 to enable interactive texture editing tools. Finally, in Chapter 7 we visit the oft-overlooked issue of filtering for the proper display of large textures, with a particular focus on normal maps.

CHAPTER 2

EXAMPLE-DRIVEN TEXTURE MODELS

Textures are noteworthy in that their repetition can be exploited for more compact representations, and ultimately for automated generation. There is a rich history of methods for modeling and synthesizing textures; we focus here particularly on the data-driven class of methods. These approaches attempt to model and recreate new texture images to resemble a given input *exemplar* image, and have received significant attention in recent years for their simplicity and output quality. We organize the discussion of this research according to the underlying texture models.

2.1 MARKOV RANDOM FIELD

Arguably the most successful framework for texture description has been the Markov random field (MRF) [Cross and Jain, 1983] model, which maps a given texture image to an undirected graph. The vertices of the graph each represent a given pixel value as a random variable, and—together with the edges—satisfy the following properties:

Markovianity — the conditional probability distribution of a pixel value, x, depends only on the adjacent values in the graph, $N(x)$. That is,

$$p(x|\text{all other pixels}) = p(x|N(x)). \tag{2.1}$$

Locality — vertices in the graph are adjacent if and only if their corresponding pixels are spatially close to each other in the texture plane.

Stationarity — the conditional distribution in (2.1) is the same for all pixels.

Put simply, a given pixel's value should depend only on the values of those pixels within its local window; furthermore, this dependence should remain constant for all possible windows in the texture. The formulation captures in analytical terms the intuition that features (pixel neighborhoods) in the exemplar should occur with the same regularity in the output as they do in the output, and furthermore that this requirement is sufficient to synthesize plausible textures. In the context of synthesis, the general approach is: to interpret the input as an MRF; to model–either explicitly or implicitly–the function $p(x|N(x))$; and finally to generate an output satisfying both this function and the MRF properties. There have been a number of works either based directly on or rooted heavily in this theoretical setting, which we examine below.

Direct modeling Early MRF-based synthesis methods attempt to directly model the conditional probability in Equation 2.1 [Popat and Picard, 1993; Zhu and Mumford, 1998; Paget, 2004]. These methods suffer chiefly from two drawbacks. First, they are typically slow to run, as Equation 2.1 must be represented a high-dimensional vector space; this incurs the so-called "curse of dimensionality". Perhaps more discouragingly, the quality of their synthesis results have in large part been completely superseded by that of later methods; this is not surprising, as these methods are focused more on finding general analytic representations than on producing high-quality synthesized results.

Pixel copying Towards the aim of synthesis, a number of methods have utilized a key observation: if the goal is simply to generate a plausible output, it is not necessary to parameterize the input at all! For a given pixel, x, we can approximate the conditional probability in Equation 2.1 simply by locating exemplar pixels with similar neighborhoods. It follows from stationarity that these pixels were drawn from the same distribution, so it is reasonable to assign the value of x from among these best-match choices. Indeed, this simple "pixel copying" approach has shown to be quite effective in generating realistic textures [Efros and Leung, 1999; Wei and Levoy, 2000].

In the limit, this approach amounts to minimizing the error between a given output

pixel neighborhood and its closest-matching neighborhood in the exemplar. A number of recent methods formalize this intuition as a global optimization problem over the entire output [Kwatra *et al.*, 2005; Lefebvre and Hoppe, 2005].

Patch copying A family of patch copying, or quilting, methods [Efros and Freeman, 2001; Wu and Yu, 2004; Kwatra *et al.*, 2003] seek to generalize the pixel-copying approach. Rather than copying single pixels from the exemplar, these methods copy entire patches at a time. New patches are placed to overlap with existing texture, and—analogously with the pixel-copying strategy of finding best-matching neighborhoods—patches are selected to match as closely as possible in the overlap region.

Although only loosely based on the formal MRF model, these methods have proven to be useful in practice. Since outputs are formed by directly copying coherent regions from the exemplar, visual quality tends to be excellent within these contiguous patches. Any visible artifacts are consequently concentrated at patch seams, and are determined mainly by patch compatibility and stitching quality. Difficulties can arise, therefore, for complex textures where plausible patch arrangements may be difficult to find.

2.2 STATISTICAL MODELS

Several schemes have been proposed which replace or augment the traditional MRF model with statistical constraints on various filter responses of the output [Heeger and Bergen, 1995; Zhu and Mumford, 1998; Portilla and Simoncelli, 2000]. These methods characterize a texture by its response to a bank of feature-detecting filters, covering a range of scales. The underlying assumption of this model is that, for some appropriately selected filter bank, two texture images will be indistinguishable if they have identical response statistics over all filters. Synthesis proceeds in a series of passes, gradually coaxing the output image's filter response statistics to match those of the exemplar.

Because the strategy relies on global statistical measures, it performs best for more stochastic textures, where spatial structures are de-emphasized. There have been some efforts to address this shortcoming by also considering localized relationships between

filter responses. Portilla and Simoncelli additionally add interscale constraints to the filter response model [Portilla and Simoncelli, 2000], but they only consider correlations between immediately adjacent scales. Kopf *et al.* introduce another interesting statistically-motivated approach [Kopf *et al.*, 2007a]; their synthesis algorithm, while being primarily based on the MRF model, additionally incorporates a step to encourage preservation of global color statistics from the exemplar. Each of these additions serve to capture some structure, but they can still miss larger or more complex structural patterns.

The statistical approach is notable in that it directly takes into account the appearance of a texture at multiple scales. This is in contrast with the MRF and other models, which are typically defined only at the finest pixel scale.

2.3 OTHER MODELS

There have been several interesting texture models that do not easily fit into the previous categories.

DeBonet [1997] proposed a multiresolution texture model based on a cross-scale Markovian model. In this model, conditional probability distributions are conditioned *only* on pixels at coarser resolutions, and do not have any direct relationships with spatially neighboring features. As with statistical models, features are represented using a bank of edge-detecting filters. A strength of this model is that it can, in the limit, naturally account for potentially complex interscale correlations. However, a limitation of this model arises from the limited amount of training data available in our problem setting (typically orders of magnitude smaller than the desired output).

Texton-based models attempt to closely model the human psychovisual system [Leung and Malik, 2001; Zhu *et al.*, 2005] by isolating exactly those features which are interpreted as semantically significant. A texture image is then characterized as a global arrangement of these *textons*. This separation of appearance and structure will bear some resemblance to our hierarchical view of multiscale texture (Section 7.5.2), in that coarser exemplars can be viewed as prescribing structure while finer exemplars play a texton-like role. However, texton-centered research has tended to originate from the vision literature,

and has therefore been geared more towards analysis than synthesis. Example-driven synthesis algorithms have largely been theoretically interesting but have thus far produced unconvincing results [Dischler *et al.*, 2002; Charalampidis, 2006].

For the special case of regular and near-regular textures, Liu *et al.* [2004] showed that synthesis quality can be improved by explicitly modeling textures as deformations and relightings on a regular lattice. Similar structural priors have been imposed in the form of feature maps [Zhang *et al.*, 2003; Matusik *et al.*, 2005; Lefebvre and Hoppe, 2006], image correspondences [Risser *et al.*, 2010], or more domain-specific models such as faces [Mohammed *et al.*, 2009]. While these algorithms produce largely compelling results, and provide valuable insight, we seek a more general study.

CHAPTER 3

APPROACH AND OVERVIEW

As we have surveyed in the previous chapter, existing models and tools are not well-suited for application for application to multiscale texture. In the remainder of this thesis we will seek to identify and address the challenges brought about by the multiscale setting.

Approach Our work is motivated chiefly by real-world limitations in production practice (Chapter 1). Therefore, the foremost consideration in this research is ultimately the usability of our methods. Within the bounds of correctness, our algorithms favor computational efficiency and ease of implementation. Likewise, we seek naturally understandable solutions, and explain the principles behind our main engineering decisions.

Given our emphasis on usability, a natural overarching principle to our work will be to adopt and extend existing tools wherever possible. This is both a practical and ideological consideration. We seek to encourage adoption of our work; to this end, tools which can be implemented "on top" of those currently in use are clearly preferable to those which must be engineered from scratch. Likewise, given the rich selection of tools available to use, we wish to avoid reinventing the wheel. We will of course need to identify and address crucial departure points for adoption to the multiscale, but as a whole we consider it a strength of our work that it flows naturally from well-studied concepts.

Overview We begin with an investigation into methods for authoring multiscale textures. A key limitation of existing synthesis tools lies in the input representation itself;

we address this in Chapter 4 with the introduction of the exemplar graph. This powerful image-based data structure enables compact specification of multiscale textures, and will serve as a crucial object for our research. In Chapter 5 we present an algorithm to synthesize outputs of arbitrary size and resolution from an exemplar graph input. Our method extends a popular single-scale synthesis scheme, and demonstrates that qualitatively new multiscale results can be achieved through a perhaps surprisingly small number of principled modifications.

Building upon our framework, we examine in Chapter 6 mechanisms to allow interactive control of the synthesis process. In seeking this goal we develop a number of optimizations to key algorithm components. These enhancements allow us then to perform novel editing operations such as the real-time modification of exemplars and globalization of local edits.

Lastly, in Chapter 7 we address the long-standing problem of filtering normal maps for display. In forming our solutions we develop a new convolution-based theory of normal mapping. This theory yields several immediate consequences: it generalizes many previous works in normal filtering; brings normal mapping within the wider umbrella of frequency-domain rendering methods; and enables the development of new techniques for accurate filtering and display.

Part II

AUTHORING MULTISCALE TEXTURES

CHAPTER 4

THE EXEMPLAR GRAPH

Before we can even begin to address the algorithms for multiscale textures, we must first address the more fundamental issue of *representation*. To illustrate this problem, recall the maplike texture introduced in Chapter 1 (Figure 7.6). In a traditional example-based synthesis scheme, textures must be specified in a single exemplar image large enough to contain the coarsest features, but with enough resolution to depict the finest—in this example, such an exemplar would be on the order of $16k \times 16k$ pixels. This is both wasteful and impractical, as there is much repeated featural content (*e.g.*, vast expanses of ocean or green land) that can be summarized in a more compact form. This is the observation underlying our *exemplar graph* representation.

Figure 4.1, right shows an exemplar graph describing this same texture. In this graph, each exemplar need only be large enough (in resolution) to faithfully capture those features that characterize a feature at a particular spatial scale. The graph arrows relate structures of differing scale: the head of an arrow points to an upsampled feature present somewhere on its tail, and the label on the arrow gives the relative scale between the exemplars. This formal decoupling of feature size and image resolution allows us to represent large textures far more efficiently; in this case, our entire planet-like structure was synthesized from just eleven 256×256 exemplars.

Beyond this significant quantitative gain, the graph representation enables *qualitatively* new types of inputs. In particular, *loops* in the exemplar graph represent an infinitely-

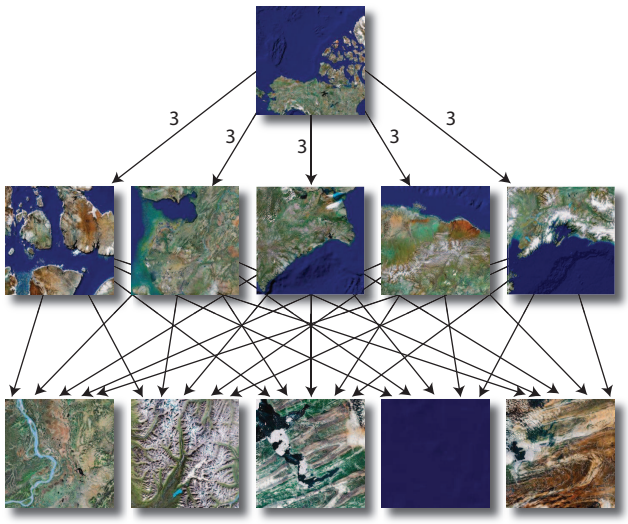


Figure 4.1: **The exemplar graph.** A desired multiscale texture *(left)* can be represented as a graph containing a set of small exemplars and associated scaling relationships *(right)*.

detailed, self-similar texture. They will enable our synthesis scheme (Chapter 5) to transform a finite resolution input into an infinite resolution output, that can be navigated by unbounded zooming and panning. Loops make the exemplar graph fundamentally more expressive than a single exemplar, since a single exemplar (of large but finite resolution) cannot allow for infinite levels of detail. By using graphs of exemplars, we take one step toward enjoying the benefits typically associated to procedural methods [Perlin, 1985; Ebert *et al.*, 2003]. At the same time, we allow for synthesis in those settings (*e.g.*, acquired data, artistic design) where a precise mathematical formulation is not readily available.

We give a more formal definition of the exemplar graph in the next section, followed by a discussion of potential problems one could encounter in working with it.

4.1 DEFINITION

The exemplar graph, $(\mathbb{V}, \mathbb{E})$, is a reflexive, directed, weighted graph, whose vertices are the exemplars, $\mathbb{V} = \{E^0, E^1, \ldots\}$, and whose edges, $\mathbb{E}$, denote similarity relations between exemplars. The *root*, E^0, serves as the coarsest-level starting point for synthesis. We fix the spatial units by declaring that root texels have unit diameter. For ease of notation,

our exposition assumes that all exemplars have resolution $m \times m$ (where $m = 2^L$), but the formulation can easily be generalized to exemplars of arbitrary size.

Figure 4.2a shows a simple graph with three exemplars. An edge, $(i, j, r) \in \mathbb{E}$, emanates from a *source* exemplar, E^i, and points to a *destination* exemplar, E^j, and carries an associated *similarity relation* r. In this thesis we consider only scaling relations, which we represent by a nonnegative integer r such that 2^r is the spatial scale of the source *relative* to the destination. For example, in Figure 4.2a the edge $(0, 1, 2)$ denotes a transition from E^0 to E^1 along with the interpretation that the diameter of a pixel in E^0 is four (2^2) times the diameter of a pixel in E^1. Likewise, pixels in E^2 are eight times smaller than those of the root. The reflexive edge $(2, 2, 1)$ indicates that E^2 is similar to a $2\times$ scaling of *itself*. Finally, since exemplars are self-similar, every exemplar has an implicit self-loop (not shown in our figures) with $r = 0$.

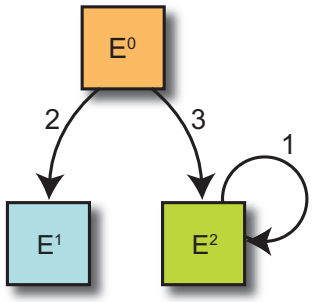


Figure 4.2: **A simple graph.** A simple exemplar graph containing three exemplars.

We do not restrict the *destination* of an edge; in particular, we permit arbitrary networks including loops (*e.g.*, the self-loop of E^2 in Figure 4.2). We do, however, require r to be less than some maximum value r_{max}; this ensures sufficient overlap between source and destination scales, as this is required to reconstruct intermediate scales.

4.2 INCONSISTENCY

With the increased expressive power of exemplar graphs comes an added caveat: the implicit information that the graph gives about the texture function may contain contradictions. This problem of *inconsistency* arise because an exemplar graph can contain arbitrary images in arbitrary arrangement. Consider, for example, the exemplar graph in Figure 4.3, which prescribes a rainbow-stripe pattern at an $8\times$ coarser scale relative to a black-and-white texture. Such a relation is clearly inconsistent, as no combination of downsampled neighborhoods in the greyscale image can reproduce the colorful appear-

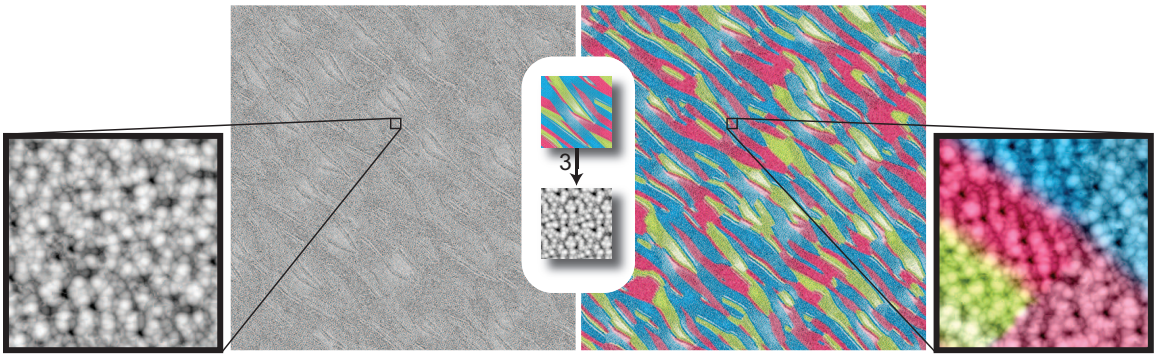


Figure 4.3: **Inconsistency correction.** An exemplar graph *(middle)* may include inconsistent relationships (edge from rainbow-streaked to grey blobby texture). Neighborhoods in the finer (grey) exemplar provide poor matches for those in the coarse (striped) exemplar *(left)*. Our proposed inconsistency correction scheme *(right)* repairs this problem by maintaining a color transfer function at each synthesis texel Section 5.3.

ance. Such contradictions do not exist in single-exemplar setting, where features of all scales are encoded in a single image; our treatment of exemplar graphs must therefore include a discussion of consistency. We list here several possible approaches.

Consistency by convention One could simply restrict the space of allowable inputs to include only strictly consistent exemplar graphs, but this would also restrict many useful and desirable applications. We would often like to use data acquired from different sources (for instance, satellite and aerial imagery), but variations in lighting and exposure make it very hard to enforce consistency in these cases. Inconsistency handling is also desirable in that it allows greater expressive power. For example, the artist-designed exemplar graph in Figure 4.3 is inconsistent, yet can specify a pleasing outputs; were inconsistency not allowed, the same result would have required much more effort on the part of the artist.

Inconsistency correction One possible corrective approach is to attempt to reconcile inconsistencies at synthesis time. We will present such a strategy as a component of our synthesis algorithm (Section 5.3). In our scheme, we establish the convention that the texture prescribed by coarser exemplars acts as a prior for the appearance of finer levels.

Figure 4.3 (right) demonstrates a result employing our inconsistency correction method. Note that we are able to compensate for the color variations between exemplars, adjusting finer-level texels to match those encountered at coarser levels in the synthesis.

Although we chose a "coarse-to-fine" convention, other conventions are also possible. In fact, this choice will have drastic implications on the corrected result. For instance, the left-side result in Figure 4.3 was generated under the inverse rule: that the coarse output appearance should be dictated by fine-level exemplars. While our approach gives (in our opinion) superior results in this comparison, we note that other inconsistency correction strategies are possible and may be an area for future exploration.

Consistentization An interesting middle-of-the-road approach that has not yet been explored is *consistentization*, wherein arbitrary inputs are allowed, and through a preprocess adjusted to form consistent graphs. This process has two potential use cases:

- Given a very large (multiscale) exemplar, decompose it to find a plausible exemplar graph. Since the input is a single image, it should be possible to extract a completely consistent graph. This process can be considered a multiscale analog (or perhaps extension) to inverse texture synthesis [Wei *et al.*, 2008], which attempts to reduce an inhomogeneous texture to a single exemplar.

- Given an arbitrary exemplar graph, modify its component exemplars to be more consistent with the prescribed scale relationships.

Pyramid optimization The preceding approaches assume a single, self-consistent image as the final rendered result. However, some settings (such as online map imagery) allow a relaxation of this assumption wherein the desired output is a pyramid of (potentially inconsistent) images. For such cases, it has been shown [Han and Hoppe, 2010] that it is possible to efficiently produce image pyramids of nearly-optimal interscale visual continuity. While this strategy has yet to be combined with texture synthesis methods, we point out such an investigation as potentially interesting future work.

CHAPTER 5

SYNTHESIS

Given the exemplar graph representation of the preceding chapter, we can now begin to develop useful tools for multiscale textures. Chief among these is the application of synthesis—that is, generating novel instances of a prescribed input texture. Our input we will be an (possibly inconsistent) exemplar graph, and our output will be a prescribed window of a deterministic, conceptually infinite texture image.

Although our input form presents a new problem setting, we fortunately have the benefit of a wealth of example-based synthesis tools and methods to draw upon. One strength of our framework is that it can directly leverage these existing techniques. In particular, we build on the method of Lefebvre and Hoppe [2005], whose parallel hierarchical synthesis approach provides a natural starting point for our algorithm. We show the insights needed to bridge the gap between conventional and multiscale hierarchical texture synthesis (Section 5.2), and furthermore demonstrate optimizations to enable GPU implementation (Section 5.4).

Our CPU and GPU implementations handle general graphs with arbitrary connectivity, including multiple loops, as evident in numerous examples derived from both user-designed textures and real-world data. Our algorithms can generate gigapixel-sized images exhibiting different features at all scales (*e.g.*, Figures 4.1, 5.4, 5.5, 5.6). Alternatively, they can render small windows of the multiscale texture at a given spatial position and scale, and even support pans and zooms into infinite resolution textures (Figure 5.3).

5.1 RELATED WORK

Our work builds on recent literature in texture synthesis, and in particular hierarchical and parallel example-based synthesis.

Texture synthesis A great deal of recent work synthesizes texture using either parametric [Heeger and Bergen, 1995; Portilla and Simoncelli, 2000], non-parametric [De Bonet, 1997; Efros and Leung, 1999; Wei and Levoy, 2000], or patch-based [Praun *et al.*, 2000; Efros and Freeman, 2001; Liang *et al.*, 2001; Kwatra *et al.*, 2003] approaches. Using only a single exemplar, these methods capture only a limited range of scales.

Hierarchical texture synthesis Hierarchical methods synthesize textures from a single exemplar whose features span varying spatial frequencies [Popat and Picard, 1993; Heeger and Bergen, 1995; Wei and Levoy, 2000]. A hierarchical method synthesizes in a coarse-to-fine manner, establishing the positions of coarse features and refining to add finer ones. This general approach serves as a natural starting point for our work.

Parallel texture synthesis Since multiscale textures are typically very large, our work incorporates ideas from parallel synthesis [Wei and Levoy, 2002; Lefebvre and Hoppe, 2005] to deterministically synthesize an arbitrary texture window at any scale. This avoids explicitly rendering to the finest available scale—in fact, recursive exemplar graphs have *no* finest scale!

Multiple exemplars and scales Several existing works employ multiple exemplars, but these methods assume equal scale across all inputs [Heeger and Bergen, 1995; Bar-Joseph *et al.*, 2001; Wei, 2002; Zalesny *et al.*, 2005; Matusik *et al.*, 2005]. Others take multiple scales into account, either explicitly [Tonietto and Walter, 2002] or in the form of local warps [Zhang *et al.*, 2003], but they do not consider scale relationships between exemplars.

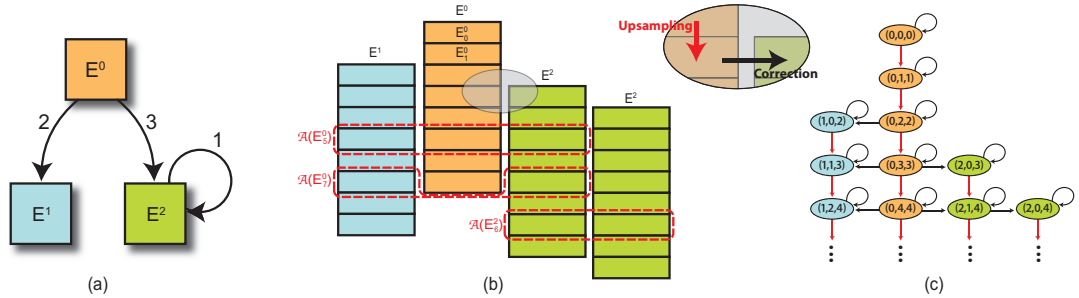


Figure 5.1: **Data structures.** *(a)* A simple exemplar graph (previously seen in Figure 4.2). *(b)* Upon computing the Gaussian stacks for each exemplar in the graph, we call those stack levels with equivalent scale *admissible candidates* of one another. To guide the synthesis process towards higher-resolution exemplars, the finest stack levels are considered inadmissible. *(c)* The superexemplar expansion of the graph shown at left. Nodes correspond to stack levels, red edges to upsampling steps, and black edges to correction passes. Node labels give (respectively) exemplar index, stack level, and *red depth*; this last quantity will be used to aid in exemplar graph analysis (Section 5.4).

5.2 MULTISCALE TEXTURE SYNTHESIS

A *graph* of exemplars opens the door to far more expressive, yet economical, design of textures. The question we address below is how to enjoy the benefits of the graph representation with a minimal set of changes to an existing hierarchical approach. Specifically, we extend the parallel, hierarchical approach of Lefebvre and Hoppe [2005], and adopt their notation where applicable.

5.2.1 DATA STRUCTURES

Adopting the traditional hierarchical approach, we build an image pyramid $S_0, S_1, \ldots, S_T$, in a coarse-to-fine order, where T depends on our desired output image size. The images are not represented by color values, but rather store coordinates, $S_t[p] = (i, k, u)$, of some stack level texel, $E_k^i[u]$. Progressing in a coarse-to-fine manner, each level S_t is generated by (1) upsampling the coordinates of S_{t-1}, (2) jittering these coordinates to introduce spatially-deterministic randomness, and then (3) locally correcting pixel neighborhoods to restore a coherent structure.

5.2.1.1 GAUSSIAN STACKS

We associate to each exemplar, E^i, its Gaussian stack, $E^i_0, E^i_1, \ldots, E^i_L$ [Lefebvre and Hoppe, 2005]. Each *stack level*, E^i_k, is an $m \times m$ image obtained by filtering the full-resolution exemplar image with a Gaussian kernel of radius 2^{L-k}. Figure 5.1b shows the Gaussian stacks associated with the exemplar graph in Figure 5.1a, positioned to show their relative scales (E^2 is shown twice to reflect its self-similarity relation). The stacks pictured are eight levels tall, corresponding to an exemplar size of 128 ($L = 7$).

5.2.1.2 ADMISSIBLE CANDIDATES

In the single-exemplar setting, neighborhood matching (Section 5.2.2.3) operates naturally on neighborhoods chosen from the same stack level as the source texel. The multiscale setting, however, requires us to consider neighborhoods from multiple candidate stack levels, and–in the presence of loops–possibly even from multiple levels within each exemplar.

The *admissible candidates* for stack level E^i_k,

$$\mathcal{A}(E^i_k) = \{\, E^j_l \mid \exists\, (i, j, k-l) \in \mathbb{E},\ 0 \le l < L \,\}\ ,$$

are determined by the exemplar graph edges emanating from E^i, and their associated scaling relations. For example, the sets of admissible candidates for three different stack levels are shown with dashed lines in Figure 5.1b. The set $\mathcal{A}(E^0_5)$ contains E^0_5, E^1_3, and E^2_2, since links $(0,0,0)$, $(0,1,2)$, and $(0,2,3)$ exist in the exemplar graph. Notice that E^2_1 is *not* admissible, as there is no link $(0,2,4)$. The finest levels of each stack (E^0_7, for example) are not admissible candidates; this is to enforce that correction (see Section 5.2.2.3) will progress to finer scales and not get "stuck" on a given exemplar. Finally, exemplar graph loops (such as the reflexive edge at E^2) can result in stack levels with candidates from the same exemplar, *e.g.*, $E^2_5 \in \mathcal{A}(E^2_6)$.

5.2.1.3 MULTISCALE CONSIDERATIONS

When using Gaussian stacks one must be careful to consider the *physical scale* of a referenced texel relative to the current synthesis level. We use $h_k = 2^{L-k}$ to denote the regular

spacing of a texel in level k of a given stack. In our framework, synthesis pixels are not "synchronized"; each synthesized pixel may point to a different exemplar, and to any level of its Gaussian stack. Therefore, whereas Lefebvre and Hoppe [2005] use a single spacing parameter h_l for each synthesis level, our spacing must be accounted for on a *per-pixel* basis since each pixel can have a unique relative scale. Additionally, our correction step must also take into account the presence of multiple exemplars. When finding a matching neighborhood for a given pixel, we search within *all* admissible candidate levels (Section 5.2.1.2).

The images shown in this chapter can be on the order of gigapixels; building and maintaining a synthesis pyramid of this size would be cumbersome and impractical. Rather, we exploit the spatial determinism of the parallel approach to generate smaller windows of the overall finest-scale texture and tile them offline. Alternatively, since we can interpret *any* scale as being the output image resolution, we can generate zooming animations (such as Figure 5.3) in real time, with finer resolutions being rendered as needed.

5.2.2 ALGORITHM

5.2.2.1 UPSAMPLING

We refine each pixel in S_{t-1} to form a coherent 2×2 patch in S_t by upsampling its coordinates. Intuitively, pixels in the upsampled image will point to the same exemplar as their parent pixels, but will move to the next-finer Gaussian stack level. Using $(i, k, u) = S_{t-1}[p]$, the upsampled patch is defined by

$$S_t\left[2p + \Delta + \left(\tfrac{1}{2}, \tfrac{1}{2}\right)\right] \coloneqq \left(i,\, k+1,\, u + \lfloor h_k \Delta \rfloor \pmod m\right) \ ,$$

where $\Delta \in \left\{\left(\pm\frac{1}{2}, \pm\frac{1}{2}\right)\right\}$.

5.2.2.2 JITTER

Next, we jitter the coordinates. Using $(i, k, u) = S_t[p]$, the jittered pixels are

$$S_t[p] \coloneqq \left(i, k, u + J_t(p) \pmod m\right) \ , \text{ where } J_t(p) = \left\lfloor h_k \mathcal{H}(p) \rho_t + \left(\tfrac{1}{2}, \tfrac{1}{2}\right) \right\rfloor .$$

This jittering step directly follows that of Lefebvre and Hoppe [2005], and we use the hash function, $\mathcal{H}$, and the level-dependent randomness coefficient, $\rho_t \in [0, 1]$, defined therein.

5.2.2.3 CORRECTION

For each synthesized pixel, $S_t[p] = (i, k, u)$, the correction step seeks among all admissible stack levels, $E_l^j \in \mathcal{A}(E_k^i)$, a texel $E_l^j[v]$, whose local 5×5 exemplar neighborhood best matches the local 5×5 synthesis neighborhood of $S_t[p]$. Formally,

> $E_l^j[v]$ is the minimizer of the error functional
>
> $$\sum_{\Delta \in \{-2 \ldots +2\}^2} \left\| {}^*S_t[p + \Delta] - E_l^j[v + \Delta h_l] \right\|^2 \tag{5.1}$$
>
> over $E_l^j \in \mathcal{A}(E_k^i)$ and $v \in \{0 \ldots 2^L{-}1\}^2$.

Here ${}^*S_t[p]$ dereferences the texel *pointer*, $S_t[p]$, to get the stored texel color. Following Lefebvre and Hoppe [2005], we perform the computation in parallel, splitting into eight subpasses to aid convergence.

Accelerated matching To accelerate neighborhood matching, we use the *k-coherence* search algorithm [Tong *et al.*, 2002]. Given the exemplar graph, our analysis algorithm identifies for each stack level texel, $E_k^i[u]$, the exemplar texels, $E_l^j[v]$, which minimize the error functional

$$\sum_{\Delta \in \{-2 \ldots +2\}^2} \left\| E_k^i[u + \Delta h_k] - E_l^j[v + \Delta h_l] \right\|^2 \tag{5.2}$$

over $E_l^j \in \mathcal{A}(E_k^i)$ and $v \in \{0 \ldots 2^L{-}1\}^2$. We choose the K best (typically, $K = 2$) spatially dispersed candidates [Zelinka and Garland, 2002] to form the candidate set $\tilde{\mathcal{A}}(E_k^i[u])$. We then adopt *coherent synthesis* [Ashikhmin, 2001], which seeks the minimum of Equation 5.1 over the set of precomputed candidates

$$\bigcup_{d \in \{-1 \ldots 1\}^2} \tilde{\mathcal{A}}\left({}^*S_t[p + d]\right) \tag{5.3}$$

drawn from the 3×3 synthesis neighborhood; to ensure that the source and destination neighborhoods are aligned, we replace $E_l^j[v + \Delta h_l]$ by $E_l^j[v + (\Delta - d)h_l]$ in Equation 5.1.

5.3 INCONSISTENCY CORRECTION

As we outline in Section 4.2, our multiscale synthesis must address the problem of graph inconsistency.

Overview Noting the coarse-to-fine direction of hierarchical synthesis, we introduce the axiom that *the visual appearance of a coarser synthesis level constrains the visual appearance of the next finer level, and by induction, all finer synthesis levels.* Considering that a given exemplar is self-consistent by definition, it follows that inconsistencies arise only as a result of inter-exemplar transitions during the correction step. Our strategy will therefore be to describe each transition with an *appearance transfer function*, $\boldsymbol{r} : RGB \rightarrow RGB$, which captures the overall change in appearance between the source and destination stack level neighborhoods. During synthesis, we will keep a history of all transitions by maintaining a *cumulative transfer function*, $\boldsymbol{R}_t[p]$, at each synthesis pixel, $S_t[p]$. Specifically, $\boldsymbol{R}_t[p]$ is the composition of all transfer functions encountered during the synthesis of $S_t[p]$, and the rendered color of pixel $S_t[p]$ is now given by $\boldsymbol{R}_t[p](^*S_t[p])$.

To formalize these ideas, consider any transfer function that is linearly composable and invertible. In our implementation, we examined both linear ($\boldsymbol{r}(\mathbf{c}) = \mathbf{A}\mathbf{c} + \boldsymbol{b}$) and constant offset ($\boldsymbol{r}(\mathbf{c}) = \mathbf{c} + \boldsymbol{b}$) functions, and found that the latter gave good results, a compact and efficiently evaluable representation, and less numerical instability during fitting.

Analysis We will need a transfer function to describe every pixel transition that happens during the correction step. Fortunately, for all source pixels, $E_k^i[u]$, we need only consider a small number of possible destinations, namely the candidate set $E_l^j[v] \in \tilde{\mathcal{A}}(E_k^i[u])$. Consequently, our transfer functions can be computed offline for all precomputed candidates (Section 5.2.2.3).

During the candidate set precomputation (Figure 5.2a), we solve for the transfer function that best transforms the destination neighborhood to match the source neighborhood

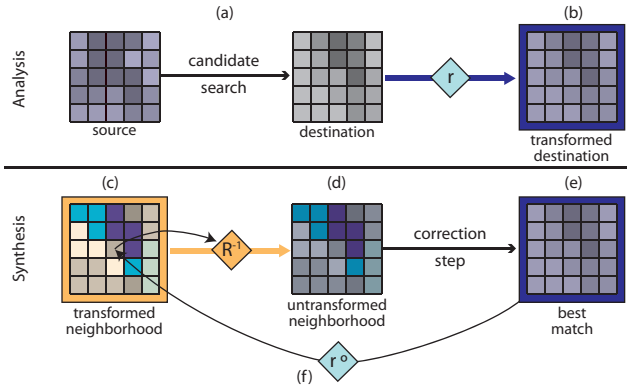


Figure 5.2: **Transfer functions.** *(a)* For every transition found during analysis, *(b)* we find a transfer function, $\boldsymbol{r}$ that minimizes color error. *(c)* At runtime, we store a cumulative transfer function, $\boldsymbol{R}$, at each synthesis pixel. Since analysis originally took place in the untransformed color space, *(d)* these transfers must be undone before performing the correction step. Finally, *(e)* we arrive at a best-match texel and its associated transfer function, which we *(f)* accumulate into the synthesis pixel by composition.

(Figure 5.2b), *i.e.*, we optimize $\boldsymbol{r}$ with respect to the metric

$$\sum_{\delta \in \{-2 \ldots +2\}^2} \left\| E_k^i[u + \delta h_k] - \boldsymbol{r}\left(E_l^j[v + \delta h_l]\right) \right\|^2 . \tag{5.4}$$

Given our choice of transfer function, $\boldsymbol{r}(\mathbf{c}) = \mathbf{c} + \boldsymbol{b}$, and the use of 5×5 neighborhoods, this yields:

$$\boldsymbol{b} = \frac{1}{25} \sum_{\delta \in \{-2 \ldots +2\}^2} \left(E_k^i[u + \delta h_k] - E_l^j[v + \delta h_l]\right) .$$

By our definition of consistency, $\boldsymbol{r}$ is the identity map ($\boldsymbol{b}$=$\mathbf{0}$) for intra-exemplar transitions.

Synthesis Recall that the correction step (Section 5.2.2.3) chooses the transition candidate that best matches the current synthesized neighborhood. We would like to match to the appearance of the *transformed* (*i.e.*, viewer-perceived) neighborhood, $\boldsymbol{R}_t[p](^*S_t[p])$ (Figure 5.2c). However, the precomputed transfer function was evaluated with respect to the actual (untransformed) texel values. Therefore, we inverse-transform the synthesis neighborhood back to the original exemplar color space used during analysis (Figure 5.2d). For our transfer functions, inversion is simply: $\boldsymbol{r}^{-1}(\mathbf{c}) = \mathbf{c} - \boldsymbol{b}$. Composing both the forward

and inverse transforms, the error functional in equation 5.1 becomes

$$\sum_{\delta \in \{-2 \ldots +2\}^2} \left\| \boldsymbol{R}_p^{-1}(\boldsymbol{R}_{p+\delta}(^*S_t[p+\delta])) - \boldsymbol{r}_v\left(E_l^j[v+\delta h_l]\right) \right\|^2 , \tag{5.5}$$

where we adopt the shorthand $\boldsymbol{R}_p = \boldsymbol{R}_t[p]$. Upon finding the best-match neighborhood (Figure 5.2e), we update the synthesis pixel by composing the associated transfer function onto $\boldsymbol{R}_t[p]$ (Figure 5.2f); for constant offset functions, composition simply amounts to adding offsets, $\boldsymbol{b}$.

During upsampling, we must propagate the cumulative transfer function to the next-finer synthesis level. We found that letting each pixel inherit its parent's transfer function (*i.e.*, a piecewise constant interpolation of $\boldsymbol{R}_{t+1}$ from $\boldsymbol{R}_t$) led to blocking artifacts. Instead, we linearly interpolate the transfer functions of the four nearest parents.

5.4 GPU OPTIMIZATION

It is often useful to have a real-time visualization of synthesized textures, *e.g.*, for tuning of jitter parameters or for application to games. As in the single-exemplar setting [Lefebvre and Hoppe, 2005], we will use principal component analysis (PCA) to make neighborhood matching more tractable on a GPU (or, alternatively, faster on a CPU). However, we first define a construction, called the *superexemplar*, that maps the exemplar graph into a form more readily treatable by existing analysis tools.

Superexemplar Formally, the superexemplar is a tree with root E_0^0 and directed red and black edges. Each vertex, $(i,k,t) \in \mathbb{V}^*$, points to a stack level, E_k^i, and its name includes a red depth counter, t. We build the superexemplar from the exemplar graph by induction:

Base step — the *root* vertex is $(0,0,0) \in \mathbb{V}^*$.

Inductive step 1 (black edge) — The admissible destinations of a correction step for stack level E_k^i are determined by the directed edges, and

associated scaling relations, of the exemplar graph:

$$\mathcal{A}^*(i,k,t) = \{\, (j,l,t) \mid \exists\, (i,j,k-l) \in \mathbb{E},\ 0 \le l \le L \,\}\ ,$$
$$(i,k,t) \in \mathbb{V}^* \longrightarrow \mathcal{A}^*(i,k,t) \subset \mathbb{V}^*\ .$$

Inductive step 2 (red edge) — The upsampling step maps a texel in stack level E^i_k to a texel in stack level E^i_{k+1}, for $k < L$:

$$(i,k,t) \in \mathbb{V}^* \longrightarrow (i,k+1,t+1) \in \mathbb{V}^*\ , \qquad \text{for } \ 0 \le k < L\ .$$

Informally, the superexemplar can be understood as (a) an unrolling of exemplar graph loops to transform the graph into a (possibly infinite) tree whose root is E^0, (b) an expansion of each exemplar graph vertex into a chain of vertices (each representing a stack level) connected by *red* edges, and (c) linking of the stack levels of corresponding exemplar graph edges with *black* edges. Figure 5.1c illustrates the superexemplar expansion of the exemplar graph shown in Figure 4.2a. Notice that red edges correspond to synthesis upsampling steps, and black edges correspond to synthesis correction steps.

The *red depth* of a vertex is the number of red edges in the unique path from the superexemplar root, E^0_0, to the vertex. This number directly corresponds to the synthesis level, t, at which the superexemplar vertex plays a role. The set of superexemplar vertices of red depth t gives us the set of stack levels that may appear at synthesis level t. This knowledge will enable us to further optimize our algorithm using PCA projection.

PCA projection We accelerate neighborhood matching (Section 5.2.2.3) by projecting the 5×5 pixel neighborhoods into a truncated 6D principal component analysis (PCA) space. However, we make two additional considerations for multiscale synthesis. First, since pixels may transition across multiple stack levels during correction, we must consider *all* stack levels that can participate at a given synthesis level. Using the superexemplar to find all levels at a given depth, we perform PCA on the set of all neighborhoods found therein to compute a suitable PCA basis.

To account for the inconsistency correction term in Equation 5.5, we first transform the target neighborhoods *before* projection into PCA space. Note that a unique transfer

function, $\boldsymbol{r}$, is associated to each candidate destination; we store alongside each candidate its transfer function and its transformed, PCA-projected neighborhood. For the GPU implementation, we also project the RGB color space down to a per-synthesis level 2D PCA space.

Texture packing Since the superexemplar provides all of the w stack levels that participate at level t, it is straightforward to map indices (i, k) at level t to one integer coordinate, $e \in [0 \dots w-1]$. This allows us to store all needed stack levels in one large $wm \times m$ texture, and to replace the u coordinate universally with $u' = me + u$.

We use one RGB texture for the stack levels $E^e(u)$; three RGBA textures for the two 6D PCA-reduced, inconsistency corrected candidate neighborhoods; and one 16-bit RGBA texture[1] to store each of the candidate links and associated transfer functions, $(\tilde{\mathcal{A}}(E^e(u)), \boldsymbol{r}_u)$. The synthesis structures $(S[p], \boldsymbol{R}[p])$ are stored in 16-bit RGBA textures.

5.5 RESULTS

We now explore the types of results enabled by our multiscale framework. Please note that the figures in this thesis have been downsampled to 150pi.

Gigapixel textures Figure 4.1 shows a 16k $\times$ 16k map texture generated using our method. The exemplar graph contains eleven exemplars of size 256×256, with scales spanning over three orders of magnitude. The large resolution of this image is able to capture features at all these scales, and allows us to evaluate the algorithm's success in synthesizing an image with spatial coherence at all scales. We faithfully recreate details at all levels, from the coarse distribution of islands to fine-level terrain details (shown in closeups). Generating such textures using existing single-exemplar methods would require an exemplar on the order of $2^{14} \times 2^{14}$ pixels, or about 400 times more data!

A similar example is shown in Figure 5.4, with the key distinction that we have disabled jitter at the coarsest levels. In this light we can interpret our method as a form of

[1] u' will generally exceed the 8-bit limit of 256.

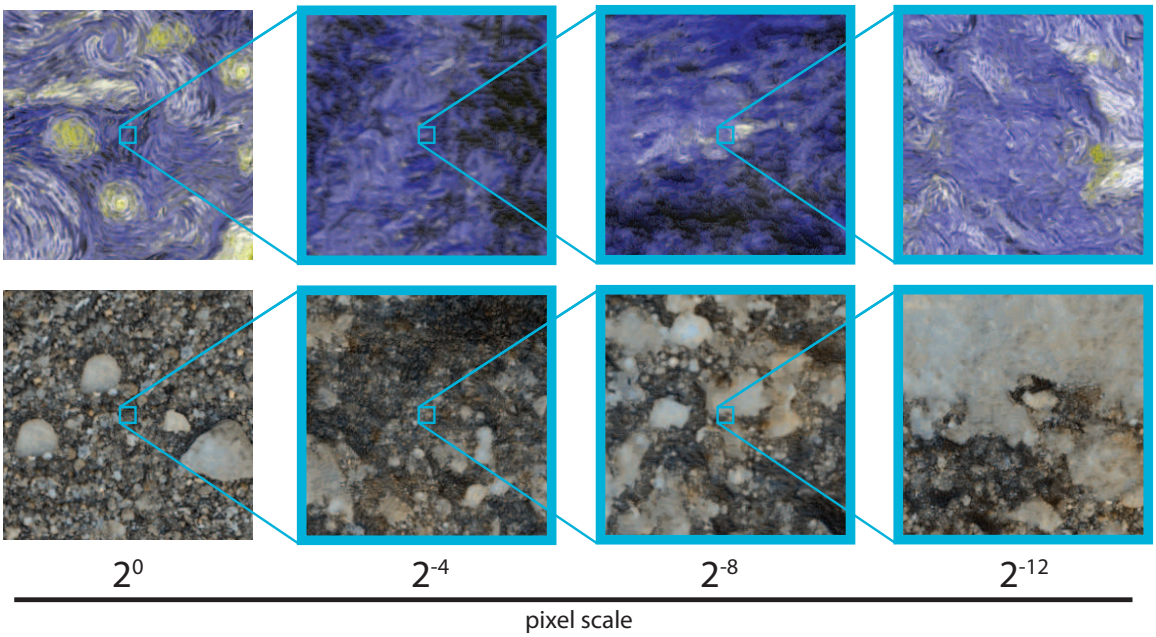


Figure 5.3: **Coherent infinite zooms.** Using a single exemplar with one reflexive edge ($r = 1$), we can specify textures with infinite detail. From left to right, each image shows a 16× zoom into the previous one. These *self-similar* textures exhibit structure at every scale, all taken from the same exemplar.

super-resolution [Freeman *et al.*, 2001; Hertzmann *et al.*, 2001]. As in previous such approaches, we employ our hierarchical texture synthesis algorithm to fill in high-resolution details on a lower-resolution image—in this case, the root exemplar (a 256 × 256 map of Japan.) However, we can deal with many more levels of detail beyond the coarse guiding image; the output image shown is again of size 16k × 16k.

Coherent infinite zooms Figure 5.3 shows frames from two infinitely zooming animations, with each image containing pixels at $1/16^{\text{th}}$ the scale of the one to its left. Notice that texture characteristics are consistently preserved across all scales. Each sequence was created using a *single* exemplar with a single self-looping edge. What we see here is an example-based approach to creating resolution-independent textures—previously attainable only through procedural methods. Furthermore, our method can utilize both artist-created (van Gogh's *The Starry Night*, top) or captured (a photograph of pebbles,

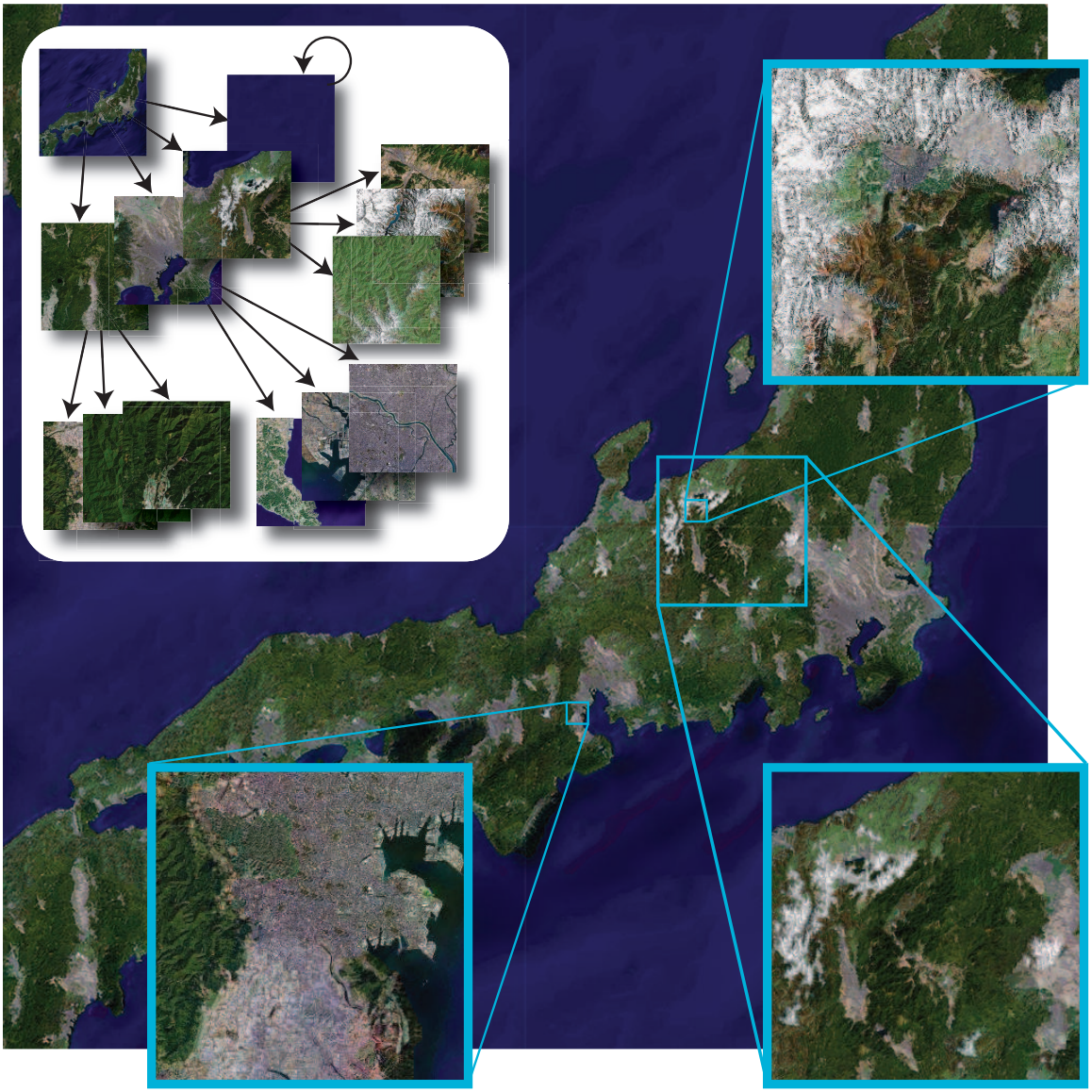


Figure 5.4: **Super-resolution.** We use fourteen exemplars and a complex topology to model a map of Japan. By disabling jitter at the coarsest levels, we "lock in" large features such as mountains and cities; these constrain the proceeding synthesis, which fills in details using the fine-scale exemplars.

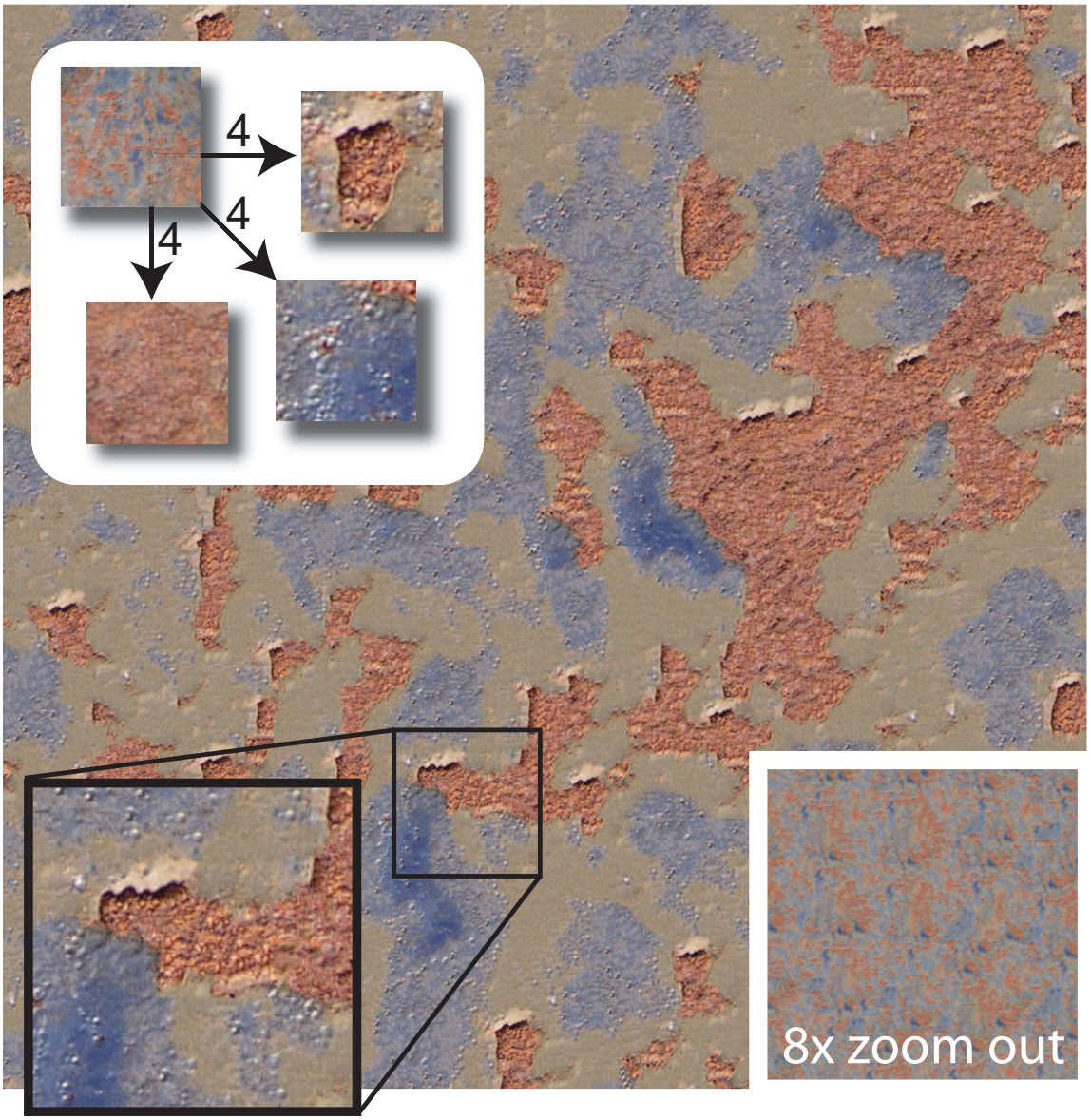


Figure 5.5: **Compact representation.** The exemplar graph used here is very small, being comprised of only four 128 × 128 exemplars; still, we are able to generate a convincing output texture several orders of magnitude larger ($8k \times 8k$, inset).

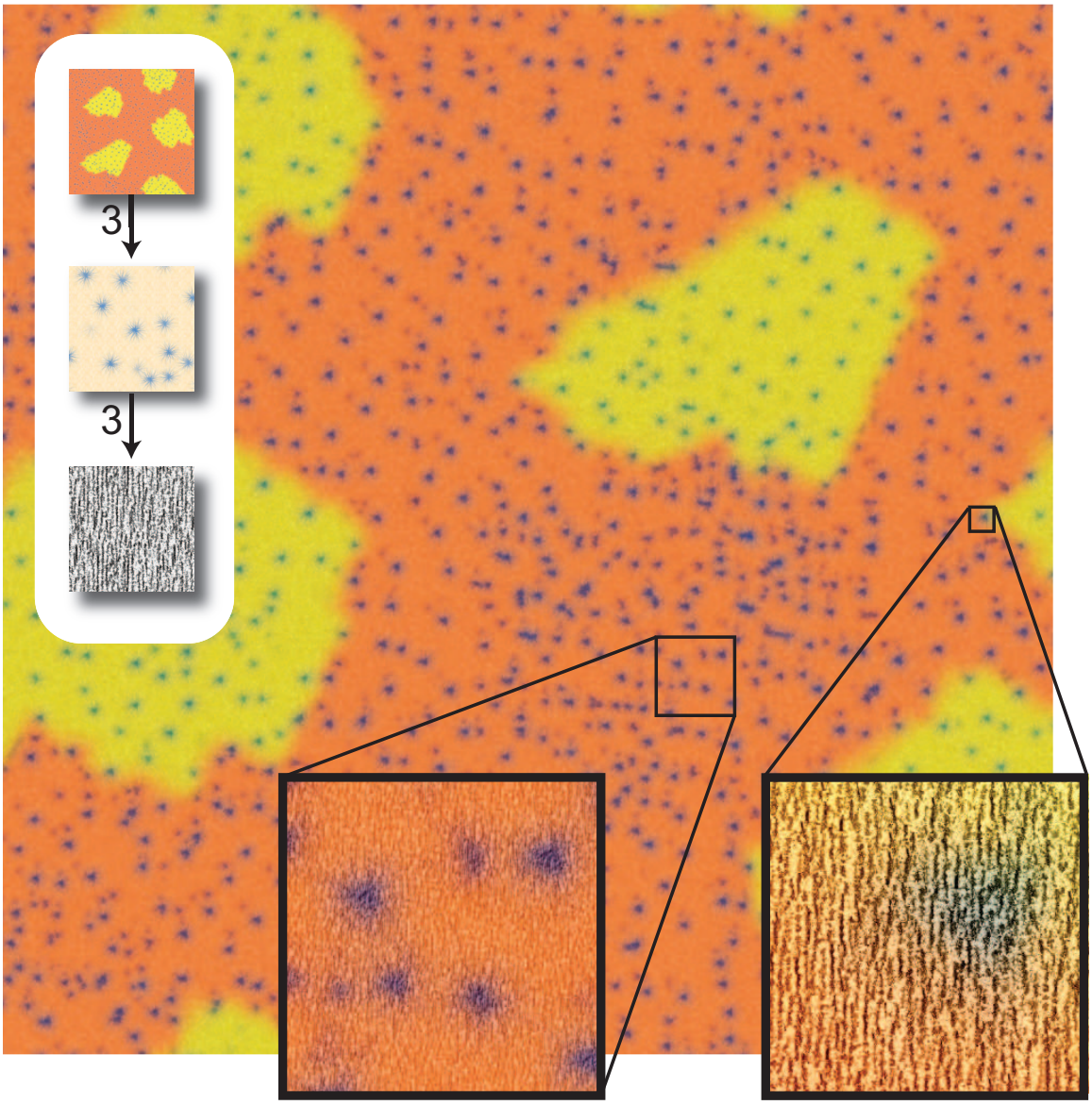


Figure 5.6: **A simple chain.** A texture created from a chain of exemplars, exhibiting unique features at three different scales. Crafted in a matter of minutes, this artist-created exemplar graph offers pleasing results that would be much harder to develop using procedural techniques. Note that inconsistencies in the input are repaired by inconsistency correction (Section 5.3).

bottom) data.

Artist controllability Finally, we show two examples that demonstrate the compact expressiveness of the exemplar graph representation. The rusted metal surface shown in Figure 5.5 was generated using just four 128×128 exemplars all taken from the same high-resolution photograph. For essentially the cost of a single 256×256 exemplar, we can produce large, aperiodic, high-resolution textures (a zoom-out is shown in the inset).

The texture in Figure 5.6 was generated from an artist-created chain of exemplars, exhibiting distinct features (yellow splotches, blue dabs of paint, and a grainy surface) at three different scales. Note that the tiny (1–2 pixel) specks in the root exemplar prescribe only the rough placement of the blue dabs, while their wispy details are contributed by the intermediate exemplar. Also notice that we achieve this result despite the largely inconsistent input. The exemplars were made in a matter of minutes, demonstrating the intuitive user control made possible by the exemplar graph; it would be much more difficult and time-consuming to create such effects using procedural methods.

CHAPTER 6

EDITING

Despite the numerous powerful tools available for unguided synthesis, there are still many situations in which user intervention is needed. For instance, an artist may need to carefully control aspects of the aspect such as final coarse feature arrangement. There will often be cases where synthesis results will exhibit slight artifacts which need to be adjusted. Or, the director may simply change his mind about the desired texture appearance. Therefore, user interaction remains a crucial part of any real-world texture authoring pipeline.

This need for interaction is growing with current production practices. With the commoditization of photographic tools and the steady increase of processing and storage capabilities, it is becoming ever easier to acquire large amounts of textural data. The problem, then, is increasingly becoming not one of synthesizing new textures from sparse input, but rather one of manipulating a wealth of available source material into the desired result. Given that the main criterion for desirability is often a subjective one (*e.g.*, the vision of an artistic director), there must be some facility to insert a human into the texture design loop.

The synthesis algorithm described in Section 5 allows real-time manipulation of some texture properties, such as perceived randomness and selective feature placement. However, these edits may prove to be insufficient to achieve the desired output. This can happen, for example, if the given exemplar graph simply does not contain the correct

features, or if the graph connectivity proves to be errant. One could modify the exemplar graph and resynthesize, but this reveals a key limitation of the scheme: it requires a potentially lengthy precomputation, and is thus not suitable for a changing input.

Our goal in this chapter, then, is to develop modifications to the analysis (*i.e.*, preprocessing) phase of the algorithm which will allow for synthesis results shown in real time. Our scheme complements the synthesis algorithm presented in Chapter 5, and therefore inherits all of its advantages of controllability and flexibility. As a part of our investigation, we will come across several technical insights (Section 6.2) which we believe are more broadly applicable.

Given our eventual goals, we lay out the following desiderata for our algorithm:

Low latency — Since our results must update in real time, we tailor our operations to return a plausible result as soon as possible. Since we employ an iterative algorithm, this will mean that we wish for the solution to be highly accurate—if not fully converged—within the first few passes.

Compatibility — Ultimately, our "target client" is the synthesis algorithm described in Chapter 5. Our algorithm should therefore be mindful of the data structures expected later on in this pipeline.

Parallelizability — Tightly coupled with these former two goals, we prefer an algorithm which can run on the GPU. This would simultaneously satisfy our desires for fast performance and compatibility with our GPU-based synthesis algorithm.

6.1 RELATED WORK

Texture editing Within the texture synthesis literature, there have been a number of proposed mechanisms for controlling the appearance of the final output.

A form of basic control exists through so-called "texture-by-numbers". In such a scheme, the artist provides as input a collection of different exemplars and a control image. The pixels of this image can either directly specify the desired output colors [Ashikhmin,

2001], or serve as an index into some (user-defined) mapping of textures [Hertzmann *et al.*, 2001]. Other methods have been developed which allow for a low-dimensional *control map* which is used to guide the synthesis process [Gu *et al.*, 2006; Wei *et al.*, 2008]. The control variable usually corresponds to some semantically meaningful image parameter (*e.g.*, age, weathering, *etc.*). While these methods provide some degree of controllability to the synthesis process, they are still not interactive. Recent work by Busto *et al.* [2010] presents a parallelizable CPU-based texture-by-numbers scheme.

There has been some research into methods for allowing direct, real-time interaction with the synthesized output. Ritter *et al.* [2006] presented an interactive texture editing tool based on the texture-by-numbers algorithm [Hertzmann *et al.*, 2001]. Their system allows for texture editing and takes into account interactions between differing textures. Ultimately, their algorithm is limited by its nearest-neighbor search algorithm and as a result is limited in the size of textures it can handle. Lefebvre and Hoppe [2005; 2006] investigated techniques to implement synthesis on the GPU, enabling interactive editing tools. However, the scope of edits offered to the user is limited. The system supports broad control over global parameters and a simple feature placement mechanism, but more sophisticated edits—and in particular, changes to the exemplar—are not permitted. This drawback is compounded by the lengthy preprocess required of the algorithm, which adds considerable latency to the overall design process. Eisenacher *et al.* [2010] propose simplifications to accelerate the preprocess, although their method is ultimately aimed at out-of-core synthesis and preview.

Image editing As textures are themselves simply a special case of general images, we can also look to the broader class of *image editing* tools. Of particular interest to us are recent methods which perform image shuffling, retargeting, refactoring, and cloning [Avidan and Shamir, 2007; Cho *et al.*, 2010; Barnes *et al.*, 2009; Farbman *et al.*, 2009; Cheng *et al.*, 2010]. These applications often require efficient subcomponents to analyze repeated content within images–certainly a relevant problem for textures.

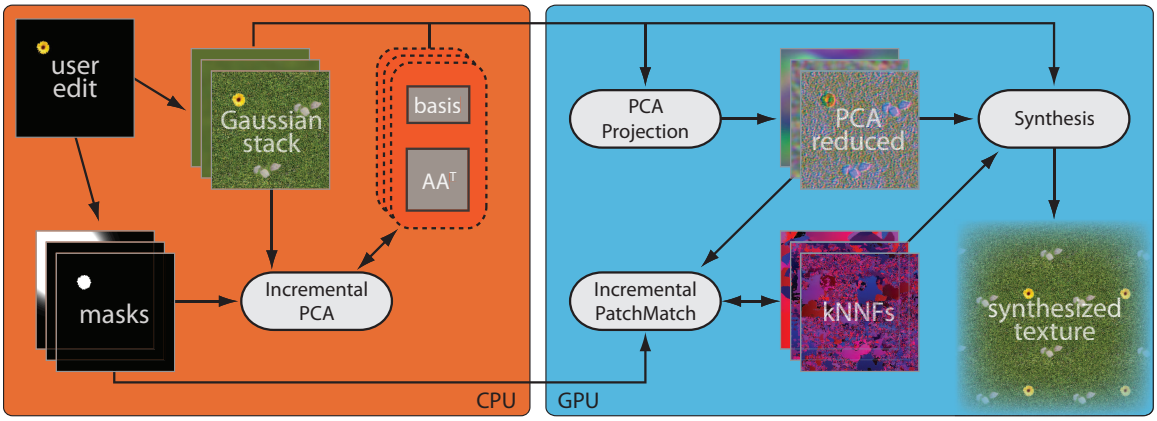


Figure 6.1: **Editing system overview.** A schematic diagram of our editing system.

Incremental PCA A key component of our algorithm is the realtime calculation of truncated PCA bases for changing images. To the best of our knowledge this specific problem has not been addressed in the literature. There exist some methods for incremental update of covariance matrices in the face of changing input data [Artač *et al.*, 2002; Dagher and Nachar, 2006], but these only treat cases in which the initial data are augmented with additional points; we are instead concerned with *replacing* the original points with updated values.

6.2 SYSTEM OVERVIEW

In an interactive editing scenario, a user is continuously changing the exemplar through a series of edits (*e.g.*, brush strokes, cut-and-paste operations, *etc*). In concept, we will need to repeat our entire precomputation each frame that the exemplar has been changed. This is clearly wasteful, however, as most edits will result in only small changes to the exemplar (and precomputation result). Our general strategy will therefore be to adopt an *incremental* scheme for efficiently updating the precomputed data structures.

Our system builds upon the work of Lefebvre and Hoppe [2005], and in particular the analysis phase. The output of this analysis should provide the following at each Gaussian stack level E_l: (1) a basis matrix for projecting patches in E_l to a lower-dimensional representation; (2) an image containing the projected patches of E_l; and (3) candidate sets

for each patch of E_l. This precomputation suffers from two bottlenecks: PCA projection of patches, and the calculation of the k-nearest matches for each patch.

Our solution (illustrated schematically in Figure 6.1) proceeds as follows. A user edit produces an updated Gaussian stack and boolean masks indicating the changed pixels at each level. These are fed as inputs into our incremental PCA algorithm (Section 6.3), which computes the updated PCA bases; having found these bases, we reproject the patches at each stack level. The resulting output proves doubly useful, as it serves as an input both to our synthesis algorithm and to the k-nearest neighbor search. This last phase of our analysis is based on the PatchMatch algorithm of Barnes *et al.* [2009; 2010]; however, we introduce additional optimizations specific to our incremental setting (Section 6.4).

We note that the candidate sets [Tong *et al.*, 2002] expected by the synthesis algorithm are essentially the same data structure as the *k-nearest neighbor field*[1] (kNNF) of Barnes *et al.* [2010]. The only difference between them is one of notational convention: the former specifies matches (candidates) as absolute image coordinates, while the latter uses coordinate offsets. For ease of exposition we will adopt the (absolute) convention of the former, but the terms should be understood to be interchangeable in this chapter.

6.3 INCREMENTAL PCA

The greatest bottleneck in performing PCA is generally the calculation of the covariance matrix $\frac{1}{N}AA^T$, where A is a $75 \times N$ matrix containing all of the (mean-centered) 5×5 neighborhoods in an N-pixel image. In the context of exemplar editing, we will often encounter situations where only relatively few pixels have changed. To efficiently handle these cases, it would be preferable to avoid recalculating the entire covariance matrix. Our goal is therefore to find an efficient operation for updating the covariance matrix for small changes to A.

[1] we use this term to distinguish the k-valued NNF from the earlier single-valued form [Barnes *et al.*, 2009]

6.3.1 DERIVATION

Note that the following derivations assume a toroidal topology for our input images. This is a reasonable and often desirable convention to follow for exemplars [Wei, 2002], and it allows for pleasing simplifications in our equations. However, there may arise cases where a toroidal input cannot be assumed. In these situations, one can either account for the image edges during both PCA and candidate search (Section 6.4), or choose to enforce nontoroidality only during candidate search. Although this latter compromise is not strictly correct, we found that it allowed for simpler computation and was a reasonable simplification for most textural images. Adoption of the more rigorous form is straightforward from our equations.

Image mean The columns of data matrix A are by construction centered about the origin; that is, it is the result $A = B - \mu$, where B is the matrix containing "raw" (uncentered) neighborhoods, and μ is a matrix whose columns are the mean neighborhood.

An arbitrary image edit will generally lead to a new mean, which we place in the columns of matrix μ'. We express the new mean-centered image as $A' = B' - \mu'$.

Changed pixels We assume that only some relatively small number of pixels (and, therefore, neighborhoods) have changed in the new image B'. Therefore, the matrix $d = B' - B$ will only have a few nonzero columns.

Refactoring For notational simplicity, we will treat the covariance matrix as AA^T (dropping the normalization term), with the understanding that the terms in the following equations will be handled properly. Since our stored quantity is AA^T, we would like to

get a recursive formulation of A' in terms of A. Using the above equations, we have:

$$\begin{aligned} A &= B - \mu \\ A' &= B' - \mu' \\ &= A - (B - \mu) + B' - \mu' \\ &= A + (\mu - \mu') + (B' - B) \\ &= A + m + d, \end{aligned}$$

where we introduce a term $m = \mu - \mu'$, and d is as defined above. Multiplying out the covariance matrix gives:

$$\begin{aligned} A' &= A + m + d \\ A'A'^T &= AA^T + Am^T + mA^T + mm^T + (A+m)d^T + d(A+m)^T + dd^T \\ &= AA^T + mm^T + (A+m)d^T + d(A+m)^T + dd^T \end{aligned}$$

Note that, following our toroidal assumption, we can eliminate the $Am^T + mA^T$ terms because the rows of m are constant, and the rows of A will always sum to 0.

For ease of computation (see next section) we will use the equivalent form:

$$A'A'^T = AA^T + mm^T + (B - \mu')d^T + d(B - \mu')^T + dd^T \tag{6.1}$$

6.3.2 COMPUTATION

Our main computational savings will come from the sparsity of matrix d, which will have zero columns for unchanged neighborhoods. We describe here how to compute the 75^2 matrix $X = dB^T$, with the understanding that the analysis can be applied to all of the "d terms" of in Equation 6.1. We propose two possible methods and discuss the tradeoffs of each.

6.3.2.1 SIMPLE METHOD

The most straightforward solution would be to perform the full multiplication $X = dB^T$, but this approach is obviously suboptimal since it disregards the sparseness of d. We therefore perform the equivalent multiplication $X = \hat{d}\hat{B}^T$, where $\hat{d}$ and $\hat{B}$ contain only the nonzero columns of d, and the corresponding columns of B, respectively. To avoid memory allocation/deallocation overhead, we preallocate large workspace matrices for $\hat{d}$ and $\hat{B}$, filling them as needed. The "real" row count is provided as a parameter to each multiplication call, assuming the use of LAPACK or a similar linear algebra library. Equation 6.1 can be performed in two LAPACK calls—a symmetric rank-1 update (`SYRK`) for dd^T, and a symmetric rank-2k update (`SYR2K`) for the B terms—plus a simple routine to add in the easily-computed matrix mm^T.

This method is simple and trivial to implement, but it does come with a few caveats. First, the use of preallocated matrices requires us to fix *a priori* n_{max}, the maximum number of neighborhoods which can be changed at once. In the event that the number of changed neighborhoods exceeds this threshold, the calculation can be performed in several passes, accumulating the result n_{max} rows at a time. Additionally, there is a small amount of redundancy incurred in the accumulation of $\hat{d}$ and $\hat{B}$, due to each changed pixel appearing in 25 different rows. We found that these issues proved to have relatively minor effects, and were overcome by the near-optimal cache coherence of this approach; however, we provide in the sequel an alternative computation which avoids temporary storage altogether.

6.3.2.2 SPACE-OPTIMIZED METHOD

A given matrix entry X_{ij} represents the sum over all image pixels p:

$$\sum_p N_{d,p}(i) * N_{B,p}(j),$$

where $N_{d,p}(i)$ denotes the pixel at location i within the neighborhood in image d centered at image location p. Consequently, for any given pixel location, we will need only to consider pixel locations within a 5-pixel radius when considering its contribution to X.

Furthermore, there is a great deal of redundancy in the computation of X. This is due to the fact that:

$$N_{d,p}(i) * N_{B,p}(j) = c * N_{d,p+\Delta}(i-\Delta) * N_{B,p+\Delta}(j-\Delta)$$

for all valid translations Δ[2]. The constant c is included to account for varying coefficients of the Gaussian kernel $G(\cdot,\cdot)$ over neighborhood locations. As a result of this relation, we can simply perform the multiplication once for each pair of pixel locations, and write it (after appropriate scaling for c) into all valid entries of X. This redundancy can be exploited up to 25 times (for $i = j$), or just once (for, i, j at opposing neighborhood corners).

The computation of matrix $d(B - \mu')^T$ proceeds according to the pseudocode in Algorithm 6.1. Note that *we do not explicitly build matrices B or d*, but rather operate directly on the original and difference images, I and $D = I' - I$, respectively. Likewise, we use only the (3D) mean pixel value $\hat{\mu}'$ rather than the full $75 \times N$ matrix μ', whose columns are $\hat{\mu}'$ multiplied by the Gaussian weighting coefficients G.

Algorithm 6.1 Space-optimized covariance computation

```
1:  for all pixel locations p do
2:    if D[p] ≠ 0 then
3:      for all pixel locations q s.t. ‖p − q‖∞ ≤ 5 do
4:        x ← D[p] ∗ (I[q] − μ̂′)
5:        for all pixel locations n s.t. ‖p − n‖∞ ≤ 2 and ‖q − n‖∞ ≤ 2 do
6:          i ← index corresponding to p − n
7:          j ← index corresponding to q − n
8:          X_ij ← X_ij + x
9:        end for
10:     end for
11:   end if
12: end for
13: X ← X ∘ H
```

Note that we do *not* perform Gaussian weighting in line 4. Since the weighting coefficients depend only on the location within X, we precompute and store 25×25 matrix H

[2]note that Δ here is a $2D$ vector while p,i, and j are scalar indices.

containing these coefficients, and perform a single component-wise multiplication at the end (line 13).

Transpose matrix $(B - \mu')d^T$ can be trivially accumulated by adding an operation $X_{ji} \leftarrow X_{ji} + x$ to the inner loop. Likewise, since dd^T follows the same access pattern, we can accumulate it the same loop structure as above; its calculation is further culled with another zero-test within the second for loop. Finally, we have $mm^T = \hat{m} * H$, where $\hat{m}$ is the change in average mean pixel value.

6.3.3 SPARSE NEIGHBORHOOD SAMPLING

The preceding algorithm describes the incremental PCA of a single image. We are dealing, however, with a full Gaussian stack—conceptually, a collection of images—within the same loop. Due to the manner of their construction, we know that a given edit will cause more changes at coarser stack levels. Indeed, even a single pixel changed at the finest level will cause the entire image to require an update at the coarsest level. This is undesirable since it requires us to perform more computation (in the extreme, a full PCA) at the coarser levels. This is especially unfortunate when we also consider that these are precisely the levels with the least "content".

We propose a simplification of the PCA basis computation: rather than considering each neighborhood from a given image, we instead collect every h_l-th neighborhood, where $h_l = 2^{L-l}$, as defined in the literature (note that this is equivalent to performing PCA on the Gaussian *pyramid*). Due to the linear construction of Gaussian stack levels, this subset of neighborhoods lies on the convex hull of the full set of neighborhoods, and the resultant PCA basis should therefore be equivalent to that computed from the full collection.

6.3.4 PCA AND NEIGHBORHOOD PROJECTION

Having computed our updated covariance matrix, it remains to find our new PCA basis and to project our neighborhoods to this new subspace. As eigenvalue computation for a problem of our size is a mostly serial process, we perform the projection on the CPU. At

this point we are able to move all proceeding computation to the GPU, beginning with the collection and projection of our image neighborhoods. Note that, although we can use sparse sampling (Section 6.3.3) when computing the basis, we need to reproject all patches in the image. The GPU-based projection is performed in the same manner as in a synthesis correction pass, with the result being directly written to a texture. The output will prove doubly useful: it will serve as an input both to our synthesis algorithm and to our optimized nearest neighbor search, which is described next.

6.4 INCREMENTAL PATCHMATCH

The PatchMatch algorithm exploits natural coherence structures within images to find nearest-matching patches. Figure 6.2 (top row) gives a schematic representation of the algorithm. In this section we describe several augmentations which make it more suitable to our algorithm goals. We construct our method components (and update existing ones) to accommodate GPU implementation and optimization.

6.4.1 NOTATION AND BACKGROUND

The kNNF, f, is a map storing the best-matching K candidates for each image patch in the input. So, the best matches for the patch centered at p are found at $f(p) = \{c_1, c_2, \ldots, c_K\}$. In following synthesis convention [Zelinka and Garland, 2002], we further prohibit candidates from being within 5 percent of the image size from each other. Furthermore, we assume an implicit identity match ($f(p)_0 = p$); the first stored candidate is therefore the best match outside of the 5 percent radius of p, followed by the next-best match not next to the first two, and so forth.

An image edit will change some number of image patches, and therefore cause our kNNF to no longer be valid. To attempt to repair the resulting errors we can run iterations of PatchMatch until the kNNF has again converged, but this proves to be suboptimal; while we will eventually find the correct solution, much of the computational effort will be spent trying to improve already converged areas. As we established at the outset,

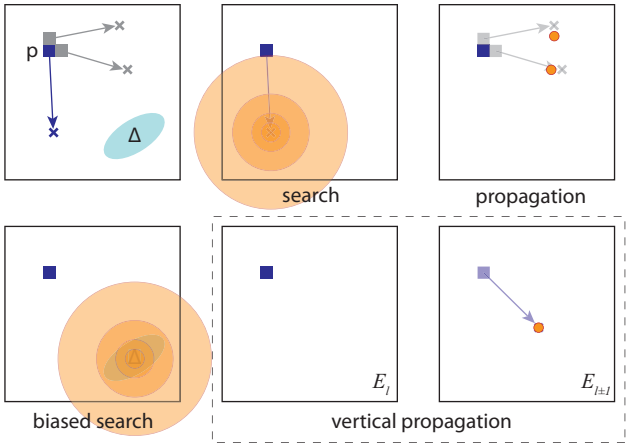


Figure 6.2: **PatchMatch phases.** We illustrate the various phases of the PatchMatch algorithm, including our additions (*bottom row*). The top left square illustrates our situation: we have a patch of interest, p (*blue square*), and its corresponding best match (*blue X*). Also shown are p's spatial neighbors and their respective matches (*grey*). A region Δ of edited pixels is also shown. Each respective phase (*other squares*) examines a small number of candidate locations (shown in orange) for best matches.

one of our main goals is to minimize latency, so we therefore seek to minimize this redundancy.

Consider a given edit, where some small subset, Δ, of patches in the image have changed, and its effect on the kNNF. We can loosely partition the resulting effects into three categories:

1. Patches in Δ. Given that these lie within the edit region, it is highly likely that their matches will require an update.

2. Patches in $f^{-1}(\Delta)$. These patches match those formerly in Δ. As the region has changed, it is likely that better matches can be found elsewhere.

3. All patches. Generally, a given edit can potentially affect *any* kNNF location if the edit causes formation of a better match.

6.4.2 RESTRICTED PASSES

The kNNF error resulting from the first two cases above is concentrated in a relatively small subset of the image domain—namely, in Δ and $f^{-1}(\Delta)$. We exploit this by running a full PatchMatch iteration *only* in these regions. In our GPU implementation, this is naturally accomplished through the use of stencil buffers and masking. Case 1 is straightforward; we have a binary mask image already available to us (provided by the UI and also used during incremental PCA), so we simply upload it to the GPU. Case 2 is slightly more involved, as the inverse kNNF f^{-1} can be of arbitrary length and is therefore not easily stored on the GPU.

To compute $f^{-1}(\Delta)$ we use a two-pass process. In the first pass, performed at every image location (*i.e.*, with masking disabled), we perform a check to see if any candidate points lie in Δ. The boolean result is written directly into the stencil buffer, which is then used as the mask for a full PatchMatch pass. Although the first pass requires us to touch every texel, the comparison operation is very cheap compared to a PatchMatch iteration. Finally, note that this step subsumes our treatment of Case 1, as Δ will necessarily be a member of $f^{-1}(\Delta)$.

6.4.3 BIASED SEARCH

As we note in Section 6.4.1, an image edit can potentially impact any part of the kNNF if it results in better matches. Since any kNNF pixel can be theoretically affected, we cannot restrict the computational domain as in Section 6.4.2; we can, however, restrict the range. Our goal is a *biased search* operation to accelerate discovery of these improvements. In the original PatchMatch algorithm, the random search phase is used to find and introduce better matches into the kNNF. Formally, a number of candidates

$$f(p) + w_i R_i$$

are examined at each pixel p, where R is a randomly generated 2D vector of length < 1, and w is a search radius. To ensure good search variety, w is varied at exponentially increasing intervals to cover a range of scales, from one pixel up to the size of the whole

image. This has the effect of "focusing" the search around a central point—namely, the current kNNF values $f(p)$.

Our biased search instead centers the search around the edit region Δ, evaluating candidates

$$\mu_\Delta + w_i R_i,$$

where μ_Δ is the centroid of Δ.

We motivate our method by examining the assumptions underlying the PatchMatch search phase. We would ideally like to test location q at a rate proportional to the likelihood that we will find a match there. In the absence of any priors, we have no indication of where in the image to find good matches, and can therefore do no better than to choose a location uniformly at random. Random search improves upon this with the implicit assumption that better matches—if they exist—are more likely to be found near the current best match. Our biased search applies this same intuition to the special situation where we are given a known edit region Δ; if better matches are to be found, they will be more likely to lie inside Δ.

Our model uses only the mean coordinate μ_Δ of the edit region. This is very simply and efficiently computed, but could be an overly simplistic model for highly anisotropic or otherwise irregular edit regions—*i.e.*, long brush strokes. We found no problem in practice, however, as our overall system runs with low enough latency to keep up with the artist's movements. Additionally, we found that keeping a exponential sampling distribution over w_i fared well at finding new matches; however, the computation might further be culled by, *e.g.*, collecting the variance σ_Δ^2 of Δ and truncating the search appropriately.

6.4.4 VERTICAL PROPAGATION

The original PatchMatch algorithm and several related methods progress in a coarse-to-fine progression to find their solutions at the finest level. This hierarchical construction is based on the fact that a low-resolution kNNF of a downsampled image often provides a good initial guess for higher-resolution solutions. However, our problem setting differs in two key aspects: first, we are interested not only in the finest-resolution kNNF, f_L,

but rather the full set of kNNFs, $\{f_l : l \in [0 \dots L]\}$, corresponding to the levels of our Gaussian stack. Furthermore, each of these of these intermediate kNNFs must be the same resolution, thus negating the efficiency gains of a pyramidal coarse-to-fine approach. We address and exploit these differences with the introduction of our *vertical propagation*[3] operation.

The idea behind vertical propagation is straightforward: at each pixel p, we consider candidates $f_{l\pm1}(p)$ from the neighboring kNNFs in the stack. Formally, we can define VerticalPropagate(f_l, E_l, p) to be:

$$\underset{\delta\in\{-1,0,1\},\, k}{\operatorname{argmin}} \; D(E_l[p], E_l[f_{l+\delta}(p)_k]).$$

Note that this operation can go in either direction—coarse-to-fine or fine-to-coarse. Indeed, we found that fine-to-coarse propagation greatly aided in overall convergence. This result agrees with our intuition; coarser-level kNNFs contain larger continuous regions, so the richer finer levels will be more likely to contribute new information than vice versa. To fully exploit this interscale flow of information, we reorder our computation: rather than solving the kNNF at each scale in sequence, we instead solve for the entire stack conceptually at once, using vertical propagation in place of upsampling. Figure 6.3 illustrates the differences between the original (left) and our updated algorithm (right), with key changes highlighted in blue.

6.5 IMPLEMENTATION AND RESULTS

We implemented our system on an 8-core, 2.00GHz machine with an nVIDIA GeForce GTX 580 GPU. Our UI displays to the artist an exemplar, which he is permitted to manipulate using basic editing tools, and a synthesized output. Additionally, a number of synthesis controls are offered, such as per-level jitter adjustment and spatial jitter control [Lefebvre and Hoppe, 2005], and the option to employ a structure-preserving jitter [Risser *et al.*, 2010]. Figure 7.8 shows representative progressions of an editing session.

[3] "vertical" here referring to the relationship between stack levels, not the image y direction

PatchMatch Algorithm

for $l \in [0 \dots L]$ **do**
  repeat
    for all locations p in f_l **do**
      $f_l(p) \leftarrow \textsc{Search}(f_l, E_l, p)$
      $f_l(p) \leftarrow \textsc{Propagate}(f_l, E_l, p)$
    end for
  until convergence
  $f_{l+1} \leftarrow \textsc{Upsample}(f_l)$
end for
return f_L

Our Algorithm

repeat
  for $l \in [0 \dots L]$ **do**
    for all locations p in f_l **do**
      $f_l(p) \leftarrow \textsc{Search}(f_l, E_l, p)$
      $f_l(p) \leftarrow \textsc{Propagate}(f_l, E_l, p)$
      $f_l(p) \leftarrow \textsc{VerticalPropagate}(f_l, E_l, p)$
    end for
  end for
until convergence
return $\{f_l : l \in [0 \dots L]\}$

Figure 6.3: **Algorithm comparison.**

The evolution of a given texture is shown in each row, driven by the artist's changes to the exemplar (shown in insets).

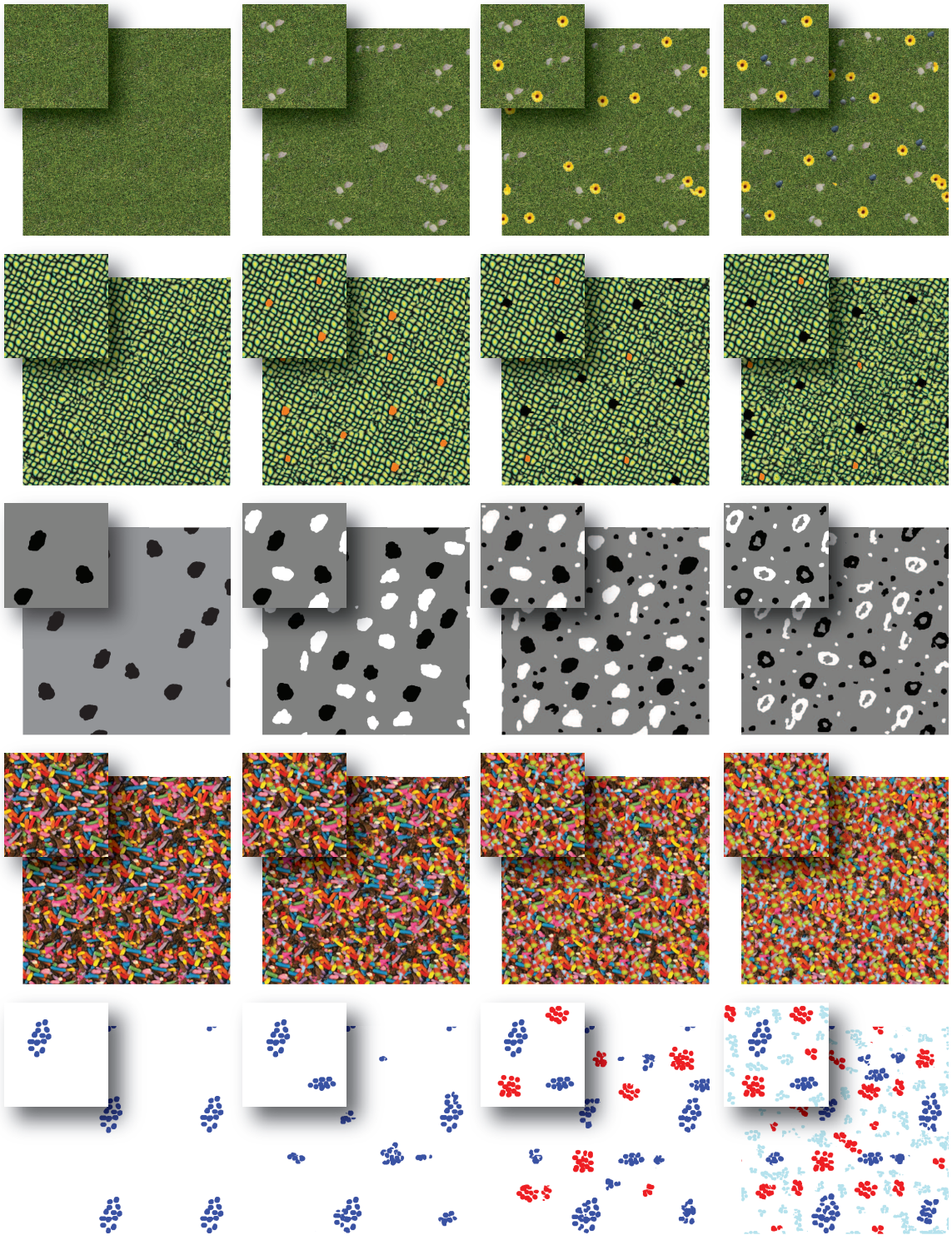


Figure 6.4: **Texture editing sessions.** We give a visual history of a number of editing sessions. In each row, an artist is progressively editing an exemplar (*insets*), with immediate feedback in the form of a synthesized texture.

Part III

FILTERING OF MULTISCALE TEXTURES

CHAPTER 7

NORMAL MAP FILTERING

By their inherently expansive nature, multiscale textures will necessarily occupy a large number of pixels. Sampled at their fullest resolutions, they can far overwhelm the capabilities of available display technology. As a result, good filtering (more specifically, downsampling) algorithms therefore become necessary in order to accurately and efficiently display them. Fortunately, there is a wealth of knowledge available in the area of image and texture filtering [Heckbert, 1989]. In particular, there has been a considerable amount of research into filtering methods for planar color images–by far the most common object for texture synthesis and editing. Indeed, tools such as MIP mapping [Williams, 1983] are so successful and ubiquitous that the filtering problem can often be all but ignored for standard color images (as—it should be noted—we have done.)

However, filtering operations still remain less well-understood for other important types of images. In this chapter we will examine one such class of texture, the *normal map*. Normal mapping (equivalently known as bump mapping [Blinn, 1978] or normal perturbation) is a simple and widely used analogue to color texture mapping wherein surface normals are specified at each texel. This allows for the expression of fine surface details, without a corresponding increase in geometric complexity. Unfortunately, as shading is not linear in the normal, the usual operations—*i.e.*, standard MIP-mapping or anisotropic filtering—cannot be applied to normal maps.

For example, consider the simple V-groove surface geometry in Figure 7.1a. In a

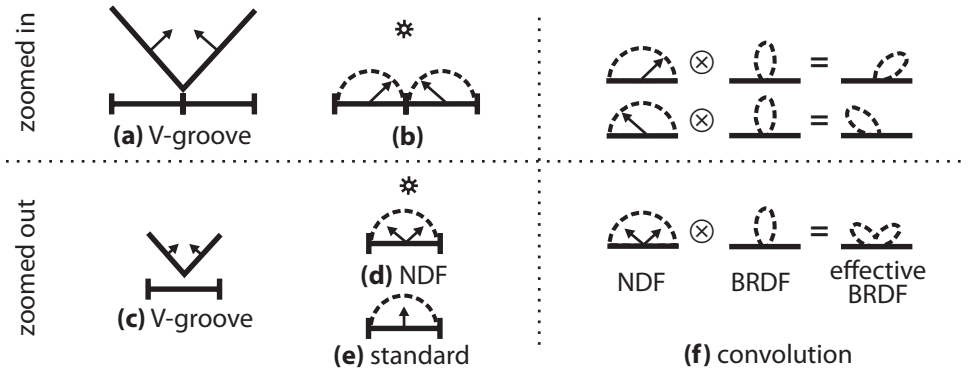


Figure 7.1: **The normal map filtering problem.** Consider a simple V-groove. Initially in closeup (a), each face is a single pixel. As we zoom out, and average into a single pixel (c), standard MIP-mapping averages the normal to an effectively flat surface (e). However, our method uses the full normal distribution function or NDF (d), that preserves the original normals. This NDF can be linearly convolved with the BRDF (f) to obtain an effective BRDF, accurate for shading.

closeup, this structure spans two pixels, each of which has distinct normals (b). As we zoom out (c), the average normal of the two sides (e) corresponds simply to a flat surface. Clearly this is incorrect, and will likely lead to an entirely different shading result than desired. As we will shortly see, what we really desire is to preserve the full normal *distribution* (d); this will allow us to achieve accurate shading through convolution with the BRDF (f).

A more complex example is shown in Figure 7.2. The inset provides both schematic (a) and diffuse-shaded (b) views of the normal map for illustration. At the top left we see a rendering at close range, where no filtering is required. However, as we zoom out (middle and bottom rows), we see that existing MIP-mapping-based realtime approaches (right two columns) result in radically different results from "ground truth" (first column)[1]. In contrast, our method (second column) is able to closely match the desired result, while rendering at interactive rates.

It has long been known qualitatively that antialiasing involves convolution of the input signal (here, the distribution of surface normals) with an appropriate low-pass

[1] "Ground truth" images are rendered using jittered supersampling (on the order of hundreds of samples per pixel) and unfiltered normal maps.

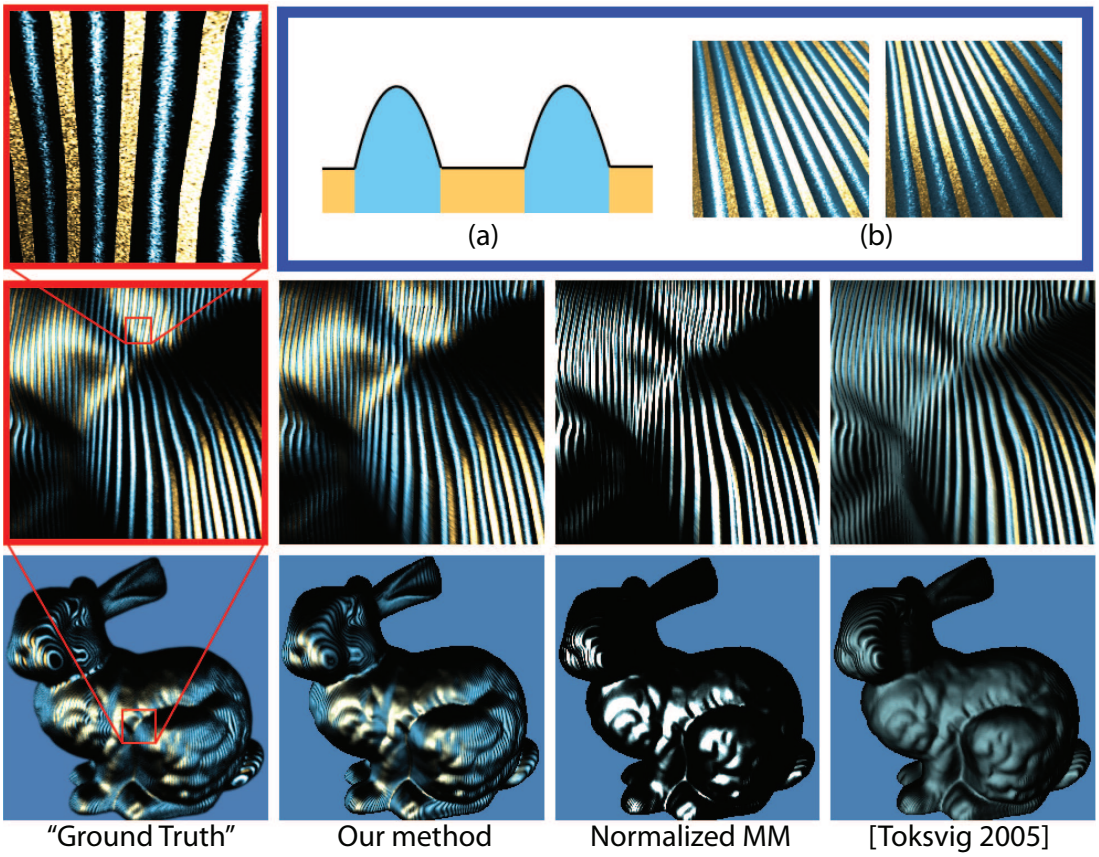


Figure 7.2: **Comparison of filtering methods.** *Top:* Closeup of the base normal map; all other methods are identical at this scale and are not shown. Schematic (a) and diffusely shaded (b) views are provided to aid in comparison/visualization. *Middle:* When we zoom out, differences emerge between our method (second column), normalized MIP-mapping (third column), and the GPU approach of Toksvig [2005] (rightmost). *Bottom:* Zooming out even further, our result is clearly more accurate than other realtime methods, and compares favorably with ground truth.

filter. The situations above illustrate the crucial limitation of using a single directional vector to represent surface normals: a usable filter is not readily available. The main insight behind our work will be to instead represent *distributions* of normals as spherical functions. In Section 7.2 and Section 7.3 we will outline a theoretical framework for normal map filtering, showing that filtering can be written as a spherical convolution of reflectance and normal distribution functions. This will lead us to a family of practical algorithms for filtering and rendering normal maps.

7.1 RELATED WORK

Normal Map Filtering Most all of the previous research into normal map filtering has–in some form or another–built upon the intuition touched upon in the preceding introduction: that the unit normal vector fails as an easily filterable representation. We can then attempt to understand previous approaches by the representations (and accompanying algorithms) that they have proposed.

Several methods approximate the distribution of normals using a single symmetric or asymmetric lobe, often employing Gaussian fitting. Olano and North [1997] modeled normal distributions consisting of a single symmetric 3D Gaussian. Asymmetric lobes—represented as 2Dcovariance matrices—have also been used to describe the distribution of normals [Schilling, 1997]. In recent work, analytical Gaussian lobe models have been used to accelerate filtering and rendering on the GPU [Toksvig, 2005; Olano and Baker, 2010]. As we explore later (Section 7.6), a single lobe is typically insufficient to capture all but the simplest normal distributions.

To capture more complex distributions, a typical strategy is to employ weighted mixtures of multiple lobes. In this vein, we find two closely-related inspirations. The work of Fournier [1992] fit mixtures of Phong lobes (up to seven per texel) using a nonlinear least-squares optimization. More recently, Tan *et al.* [2005] modeled normal distributions as mixtures of planar-projected 2DGaussians. These methods bear similarities to our lobe-based algorithm (Section 7.6), and in fact can be understood to be special cases within

our framework.

Indeed, we see our theoretical result (Section 7.3) as unifying all of the above approaches. One key result of our theory is that it illuminates the relationship between appearance (BRDF) and geometry (NDF, Section 7.3.1). Whereas prior methods required a specific rendering strategy to accompany the choice of normal representation, our formal convolution theory allows us to separate the two. Thus, once filtered, the same normal map can easily be rendered with a wide range of different BRDFs, even changing them at runtime.

Hierarchy of Representations A hierarchy of scales, with geometry transitioning to bump or normal maps, then transitioning further to BRDFs, was first proposed by Kajiya [1985]. This idea was explored further by Becker and Max [1993], although they did not focus on the filtering of any particular representation. Similarly, appearance-preserving simplification methods replace fine-scale geometry with normal and texture maps [Cohen *et al.*, 1998]. It is likely that our approach could enable continuous level of detail and antialiasing in these methods.

Convolution and Precomputed Radiance Transfer (PRT) Many of our mathematical representations and ideas derive from previous spherical convolution techniques [Basri and Jacobs, 2001; Ramamoorthi and Hanrahan, 2001]. We also build on PRT methods that have used spherical harmonics [Sloan *et al.*, 2002]. Our spherical vMF method extends zonal harmonics [Sloan *et al.*, 2005] and spherical radial basis functions [Tsai and Shih, 2006]. We also considered wavelet methods (introduced for reflectance in [Lalonde and Fournier, 1997]), but found the number of terms for an artifact-free solution too large for practical use, even with smoother wavelets.[2] Claustres *et al.* [2007] present a real-time method for wavelet-based BRDF representation and filtering on the GPU, although they do not account for normals.

We emphasize, however, that ours is not a PRT algorithm; it requires minimal precom-

[2] PRT methods can use a coarse wavelet approximation of the lighting, since it is not visualized directly, but we directly visualize NDF and BRDF.

putation and works with conventional real-time rendering techniques. Furthermore, our method rests on an explicit analytic convolution formula and uses the representations above solely for normal map filtering, not for PRT.

7.2 PRELIMINARIES

In this section, we introduce relevant notation for the reflectance equation, and BRDF and normal map representations.

The reflected light B at a surface point $\boldsymbol{x}$ with normal direction $\boldsymbol{n}$ in direction $\boldsymbol{\omega}_o$ is given by the standard reflectance equation:

$$B(\boldsymbol{x}, \boldsymbol{\omega}_o) = \int_{S^2} L(\boldsymbol{x}, \boldsymbol{\omega}_i)\rho(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)\, d\boldsymbol{\omega}_i,$$

where L is the lighting at $\boldsymbol{x}$ from incident direction $\boldsymbol{\omega}_i$, and ρ is the BRDF.

We make two modifications to this form. First, following general readability practices, the following discussion will "bake in" the cosine falloff term $(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)$ to the BRDF itself, making ρ a general transfer function rather than a BRDF in the strictest definition[3]. Also, since the BRDF is often defined in the local coordinate frame of the surface, we express this explicitly using a change of coordinates

$$B(\boldsymbol{x}, \boldsymbol{\omega}_o) = \int_{S^2} L(\boldsymbol{x}, \boldsymbol{\omega}_i)\rho(\boldsymbol{\omega}_i', \boldsymbol{\omega}_o')\, d\boldsymbol{\omega}_i, \tag{7.1}$$

where parameters $\boldsymbol{\omega}_i'$ and $\boldsymbol{\omega}_o'$ denote directions in the local frame. To find these directions, we must project or rotate the global incident and outgoing directions $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_o$ to the local tangent frame. This local frame is defined by the surface normal $\boldsymbol{n}$ and a tangent direction. As we will limit ourselves to isotropic BRDFs, the tangent direction will not be important; thus we can use any rigid rotation $R_{\boldsymbol{n}}$ mapping the normal to the z-axis [Ramamoorthi and Hanrahan, 2001],

$$\boldsymbol{\omega}_i' = R_{\boldsymbol{n}}(\boldsymbol{\omega}_i) \qquad \boldsymbol{\omega}_o' = R_{\boldsymbol{n}}(\boldsymbol{\omega}_o).$$

[3] We will continue use the term "BRDF" to refer to this new object, making clarifications as needed

Finally, although we specify an integration over the sphere S^2 of incident directions, we will mainly be concerned with rendering under a small number of point lights. The evaluation of Equation 7.1 will therefore become in practice a summation over discrete directions $\boldsymbol{\omega}_i$. We will relax this restriction, and discuss extensions to environment maps, in Section 7.6.3.

7.2.1 BRDF REPRESENTATION AND PARAMETERIZATION

Effective BRDF: We define a new function, the *effective BRDF* or transfer function that depends on the surface normal (that we denote as $\boldsymbol{n}(\boldsymbol{x})$ or simply $\boldsymbol{n}$ for clarity) as,

$$\rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \boldsymbol{n}) = \rho\left(R_{\boldsymbol{n}}(\boldsymbol{\omega}_i), R_{\boldsymbol{n}}(\boldsymbol{\omega}_o)\right),$$

allowing us to write Equation 7.1 using the global directions,

$$B(\boldsymbol{x}, \boldsymbol{\omega}_o) = \int_{S^2} L(\boldsymbol{x}, \boldsymbol{\omega}_i)\rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \boldsymbol{n}(\boldsymbol{x}))\, d\boldsymbol{\omega}_i. \tag{7.2}$$

BRDF Parameterizations: Many BRDFs can be written as

$$\rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \boldsymbol{n}) = f(\boldsymbol{n} \cdot \boldsymbol{\omega}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)), \tag{7.3}$$

where the 1D function f is radially symmetric about the shading normal $\boldsymbol{n}$, and depends on the chosen parameterization $\boldsymbol{\omega}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$ (henceforth $\boldsymbol{\omega}$). In this chapter, we focus most of our effort on these types of BRDFs, which encompass Lambertian, Blinn-Phong or microfacet half angle (like Torrance-Sparrow), and many factored and measured BRDFs.

A very common example is Lambertian reflectance, where the transfer function is simply the cosine of the incident angle, so that $\boldsymbol{\omega} = \boldsymbol{\omega}_i$, and $f(u) = \max(u, 0)$. The Blinn-Phong specular model with exponent s uses a transfer function of the form $f(u) = u^s$, with the half-angle parameterization, $\boldsymbol{\omega} = \boldsymbol{\omega}_h = \frac{\boldsymbol{\omega}_i + \boldsymbol{\omega}_o}{\|\boldsymbol{\omega}_i + \boldsymbol{\omega}_o\|}$. Measured BRDF functions $f(\boldsymbol{\omega}_h \cdot \boldsymbol{n})$ can also be used.[4]

[4] A number of recent chapters have proposed factored BRDFs for measured reflectance. [Lawrence *et al.*, 2006] uses a factorization $f(\theta_h)g(\theta_d)$, in terms of half and difference angles. The $f(\theta_h)$ term clearly fits into the framework of equation 7.3, but the BRDF now also includes a product with $g(\theta_d)$. However, θ_d does not depend on $\boldsymbol{n}$ (and g does not need to be filtered). Thus, our framework also applies to general BRDFs of the form $f(\boldsymbol{\omega} \cdot \boldsymbol{n})g(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$, where the g factor does not depend directly on $\boldsymbol{n}$.

7.2.2 NORMAL MAP REPRESENTATION AND FILTERING

Normal Map Input Representation: There are many equivalent normal map representations, including bump maps and normal offsets. For simplicity we use normal maps, parameterized on a plane, that directly specify the normal in tangent space. In the actual implementation, we perform all computations in the local tangent frame of the geometric surface; lighting and view are projected into this local frame, allowing the planar normal map to be used directly without explicit rotation. For simplicity in the proceeding discussion, the reader can therefore assume a planar underlying surface while understanding that the extension to curved 3
D geometry is straightforward.

Normal Map Filtering: In screen space, the exitant radiance or pixel color $B(\boldsymbol{x}, \boldsymbol{\omega}_o)$ at a surface location $\boldsymbol{x}$ should represent the average radiance at the N corresponding finer-level texels q:

$$\begin{aligned} B(\boldsymbol{x}, \boldsymbol{\omega}_o) &= \frac{1}{N} \sum_{q \in \boldsymbol{x}} \int_{S^2} L(\boldsymbol{x}, \boldsymbol{\omega}_i) \rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \boldsymbol{n}(q))\, d\boldsymbol{\omega}_i \\ &= \int_{S^2} L(\boldsymbol{x}, \boldsymbol{\omega}_i) \left(\frac{1}{N} \sum_{q \in \boldsymbol{x}} \rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \boldsymbol{n}(q)) \right) d\boldsymbol{\omega}_i. \end{aligned}$$

This formulation allows us to define a new effective BRDF,

$$\rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \boldsymbol{x}) = \frac{1}{N} \sum_{q \in \boldsymbol{x}} \rho \left(R_{\boldsymbol{n}(q)}(\boldsymbol{\omega}_i), R_{\boldsymbol{n}(q)}(\boldsymbol{\omega}_o) \right). \tag{7.4}$$

Note that the effective BRDF now depends implicitly on all the normals $\boldsymbol{n}(q)$ at $\boldsymbol{x}$, rather than on a single normal. At a high level, this chapter is about ways to efficiently compute and represent ρ^{eff}. To this end, the next section shows how to explicitly represent ρ^{eff} as a convolution of the original BRDF and a new function we call the NDF.

7.3 NORMAL MAPPING AS CONVOLUTION

In this section, we introduce our theoretical framework for normal map filtering as convolution. The next sections describe mathematical representations that can be used for

practical implementation.

7.3.1 NORMAL DISTRIBUTION FUNCTION

Our first step is to convert equation 7.4 into continuous form, defining

$$\rho^{\text{eff}}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o; \gamma(\cdot)) = \int_{S^2} \rho\left(R_{\boldsymbol{n}}(\boldsymbol{\omega}_i), R_{\boldsymbol{n}}(\boldsymbol{\omega}_o)\right) \gamma(\boldsymbol{n})\, d\boldsymbol{n}, \tag{7.5}$$

where $\gamma(\boldsymbol{n})$ is a new function that we introduce and define as the *normal distribution function* (NDF), and the integral is over the sphere S^2 of surface orientations. Note that a unique NDF $\gamma(\boldsymbol{n})$ exists at each surface location $\boldsymbol{x}$; for a discrete normal map, $\gamma(\boldsymbol{n})$ would simply be a sum of (spherical) delta distributions at $\boldsymbol{n}(q)$, the fine-scale normals at $\boldsymbol{x}$. Formally, $\gamma(\boldsymbol{n}) = \frac{1}{N}\sum_{q\in\boldsymbol{x}} \delta(\boldsymbol{n} - \boldsymbol{n}(q))$, as seen in Figure 7.1d. For some procedurally generated normal maps, $\gamma(\boldsymbol{n})$ may be available analytically.

7.3.2 FREQUENCY-DOMAIN ANALYSIS IN 2D

Although we will not directly use the results of this section for rendering, we can gain many insights by starting in the simpler 2D case. This "flatland" analysis is easier because the rotation operator in equation 7.5 is given simply by $R_n(\omega) = \omega + n$, yielding

$$\rho^{\text{eff}}(\omega_i, \omega_o; \gamma(\cdot)) = \int_0^{2\pi} \rho(\omega_i + n, \omega_o + n)\, \gamma(n)\, dn. \tag{7.6}$$

Significant new insight is gained by analyzing equation 7.6 in the frequency domain. Specifically, we expand in Fourier series:

$$\begin{aligned} \gamma(n) &= \sum_k \gamma_k F_k(n) \\ \rho(\omega_i + n, \omega_o + n) &= \sum_l \sum_m \rho_{lm} F_l(\omega_i + n) F_m(\omega_o + n), \end{aligned} \tag{7.7}$$

where $F_k(n)$ are the familiar Fourier basis functions $\frac{1}{\sqrt{2\pi}} e^{\mathbf{i}kn}$. Noting that $F_k(\omega + n) = \sqrt{2\pi} F_k(\omega) F_k(n)$, equations 7.6 and 7.7 can be simplified to

$$\begin{aligned} \rho^{\text{eff}}(\omega_i, \omega_o; \gamma(\cdot)) &= 2\pi \sum_{k,l,m} \gamma_k \rho_{lm} F_l(\omega_i) F_m(\omega_o) \times \\ &\qquad \int_0^{2\pi} F_k(n) F_l(n) F_m(n) dn. \end{aligned} \tag{7.8}$$

The integral above involves a triple integral of Fourier series, and we denote the corresponding tripling coefficients C_{klm}. These tripling coefficients have recently been studied [Ng *et al.*, 2004], and for Fourier series they vanish unless $k = -(l+m)$, where $C_{klm} = \frac{1}{\sqrt{2\pi}}$. Since ρ^{eff} above is already expressed in terms of $F_l(\omega_i)F_m(\omega_o)$, we can write a formula for its Fourier coefficients:

$$\rho^{\text{eff}}_{lm} = \sqrt{2\pi}\gamma_{-(l+m)}\rho_{lm}. \tag{7.9}$$

Discussion and Analogy with Convolution: Equation 7.9 gives a very simple product formula for the frequency coefficients of the effective BRDF. This is much like a convolution, where the final Fourier coefficients are a product of the Fourier coefficients of the functions being convolved (here the NDF and BRDF). However, the convolution analogy is not exact, since equation 7.8 involves a triple integral and n appears thrice in equation 7.6. In 3D, the formulae and sparsity for triple integrals in the frequency domain (especially those involving rotations) are much more complicated [Ng *et al.*, 2004]. Fortunately, many BRDFs are primarily single-variable functions $f(\boldsymbol{\omega} \cdot \boldsymbol{n})$ as in equation 7.3. In these cases, we will obtain a spherical convolution of the NDF and BRDF.

7.3.3 FREQUENCY-DOMAIN ANALYSIS IN 3D

To proceed with analyzing equation 7.5 in the 3D case, we substitute the form of the BRDF from equation 7.3. Recall in this case that the BRDF only depends on the angle between $\boldsymbol{\omega}$ and the surface normal $\boldsymbol{n}$, and is given by $f(\boldsymbol{\omega} \cdot \boldsymbol{n})$. The effective BRDF is now also only a function of $\boldsymbol{\omega}$,

$$\rho^{\text{eff}}(\boldsymbol{\omega}; \gamma(\cdot)) = \int_{S^2} f(\boldsymbol{\omega} \cdot \boldsymbol{n})\gamma(\boldsymbol{n})\, d\boldsymbol{n}. \tag{7.10}$$

Note that the initial BRDF $\rho(\cdot) = f(\boldsymbol{\omega} \cdot \boldsymbol{n})$ is symmetric about $\boldsymbol{n}$, but the final result $\rho^{\text{eff}}(\boldsymbol{\omega})$ is an arbitrary function on the sphere and is generally not symmetric.

We would like to analyze Equation 7.10 in the frequency domain, just as we did with Equation 7.6. In 3D, we must use the spherical harmonic (SH) basis functions $Y_{lm}(\cdot)$, which are the frequency domain analog to Fourier series on the unit sphere. The l index

is the frequency with $l \geq 0$, and $-l \leq m \leq l$,

$$\gamma(\boldsymbol{n}) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \gamma_{lm} Y_{lm}(\boldsymbol{n}) \qquad f(\boldsymbol{\omega} \cdot \boldsymbol{n}) = \sum_{l=0}^{\infty} f_l Y_{l0}(\boldsymbol{\omega} \cdot \boldsymbol{n})$$

$$\rho^{\text{eff}}(\boldsymbol{\omega}) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \rho^{\text{eff}}_{lm} Y_{lm}(\boldsymbol{\omega}).$$

The above is a standard function expansion, as in Fourier series. Note that the symmetric function $f(\boldsymbol{\omega} \cdot \boldsymbol{n})$ is expanded only in terms of the zonal harmonics $Y_{l0}(\cdot)$ ($m = 0$), which are radially symmetric and thus depend only on the elevation angle.

Equation 7.10 has been extensively studied in recent years, within the context of lighting-BRDF convolution for Lambertian or radially symmetric BRDFs [Basri and Jacobs, 2001; Ramamoorthi and Hanrahan, 2001]. In those works, the NDF $\gamma(\boldsymbol{n})$ is replaced by the incident lighting environment map. Since the theory is mathematically identical, we may directly use their results. Specifically, Equation 7.10 expresses a spherical convolution of the NDF $\gamma(\boldsymbol{n})$ with the BRDF filter f. In particular, there is a simple product formula in spherical harmonic coefficients, similar to the way standard convolution can be expressed as a product of Fourier coefficients,

$$\rho^{\text{eff}}_{lm} = \sqrt{\frac{4\pi}{2l+1}} f_l \gamma_{lm}.$$

Explicitly making the NDF and effective BRDF functions of a texel q, we have

$$\boxed{\rho^{\text{eff}}_{lm}(q) = \hat{\rho}_l \gamma_{lm}(q)} \qquad \hat{\rho}_l = \sqrt{\frac{4\pi}{2l+1}} f_l \ , \tag{7.11}$$

where the NDF considers all normals covered by q. While q usually corresponds to a given level and offset in a MIP-map, it can also consider more general "footprints"—we show an example with anisotropic filtering in Figure 7.3.

Generality and Supported BRDFs: The form above is accurate for all BRDFs described by Equation 7.3, including Lambertian, Blinn-Phong and measured microfacet distributions. Moreover, our results also apply when the BRDF has an additional Fresnel or $g(\theta_d)$ multiplicative factor, since θ_d (and hence g) does not depend on $\boldsymbol{n}$ and does not need to be filtered.

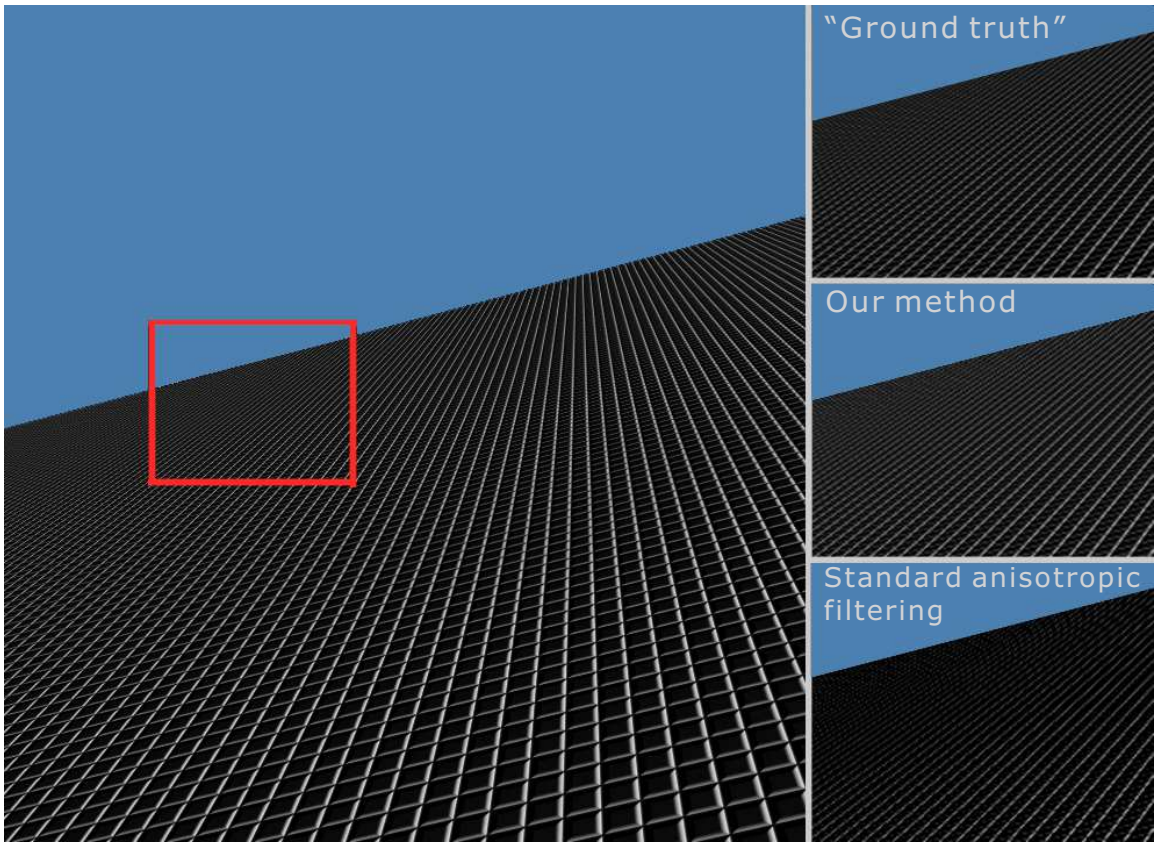


Figure 7.3: **Spherical harmonic anisotropic filtering.** Rendered under Lambertian reflection. Note the behavior for far regions of the plane. With standard normal filtering, these regions are averaged to a nearly flat surface. By contrast, our method is quite accurate, with only slight blurring in distant regions.

Note that for some specular BRDFs, we also need to multiply by the cosine of the incident angle for a full transfer function. For the spherical vMF method in Section 7.6, we address this by simply multiplying for each lobe by the cosine of the angle between light and lobe center (or effective normal). For the spherical harmonic method in Section 7.4, we simply use the MIP-mapped normals for the cosine term, since it is a relatively low-frequency effect.

7.4 SPHERICAL HARMONICS

To recap, we have as input a normal map which provides a single normal $\boldsymbol{n}(q_0)$ for each finest-level texel q_0. We also have a BRDF $\rho(\cdot) = f(\boldsymbol{\omega} \cdot \boldsymbol{n})$, with spherical harmonic coefficients $\hat{\rho}_l$. In this section, we develop a spherical harmonics-based algorithm from the final formula in equation 7.11. Later, Section 7.6 will discuss an alternative algorithm better suited for higher-frequency effective BRDFs. While the theory in the previous section is somewhat involved, the practical algorithm in this section is relatively straightforward, involving two basic steps: (1) computing the NDF spherical harmonic coefficients $\gamma_{lm}(q)$ for each (coarse-level) texel q of the normal map, and (2) rendering the final color by directly implementing equation 7.11 in a GPU pixel shader.

7.4.1 ALGORITHM

Computing NDF Coefficients: We compute a MIP-map of NDF coefficients[5], starting with the finest level normal map, and moving to coarser levels. At the finest level (denoted by subscript 0), $\gamma(q_0)$ is a delta distribution at $\boldsymbol{n}(q_0)$, i.e., $\gamma(q_0) = \delta(\boldsymbol{n} - \boldsymbol{n}(q_0))$ with corresponding spherical harmonic coefficients[6]

$$\gamma_{lm}(q_0) = Y_{lm}(\boldsymbol{n}(q_0)).$$

An important insight is that, unlike the original normals, these spherical harmonic NDF coefficients $\gamma_{lm}(q_0)$ can now correctly be *linearly* filtered or averaged for coarser levels $\gamma_{lm}(q)$. Hence, we can simply MIP-map the spherical harmonic coefficients $\gamma_{lm}(q_0)$ in the standard way, and *no non-linear fitting is required*.

---

[5] As explained in Section 7.2.2, we are operating in the local tangent frame of the geometric surface, with lighting and view projected into this frame. Thus, we do not need to explicitly consider rotations into the global frame. Note that the overall geometric surface is assumed to be locally planar (a single "geometric normal") over the region being filtered.

[6] We use the real form of the spherical harmonics, rather than the complex form, to simplify implementation. Otherwise, $\gamma_{lm}(q_0) = Y^*_{lm}(\boldsymbol{n}(q_0))$.

Rendering: Rendering requires knowing the NDF coefficients $\gamma_{lm}(q)$, the BRDF coefficients $\hat{\rho}_l$, and then applying equation 7.11. We have already computed a MIP-map of NDF coefficients. At the time of rendering, we also know the BRDF. For many analytic models, formulae for $\hat{\rho}_l$ are known [Ramamoorthi and Hanrahan, 2001]. For example, for Blinn-Phong, $\hat{\rho}_l \approx e^{-l^2/2s}$ where s is the Phong exponent. For measured reflectance, $\hat{\rho}_l$ is obtained directly by a spherical harmonic transform of $f(\boldsymbol{\omega} \cdot \boldsymbol{n})$.

Now, we can compute the spherical harmonic coefficients of the effective BRDF, per equation 7.11. Finally, to evaluate it, we must expand in terms of spherical harmonics,

$$\rho^{\text{eff}}(\boldsymbol{\omega}, q) = \sum_{l=0}^{l^*} \sum_{m=-l}^{l} \hat{\rho}_l \gamma_{lm}(q) Y_{lm}(\boldsymbol{\omega}), \tag{7.12}$$

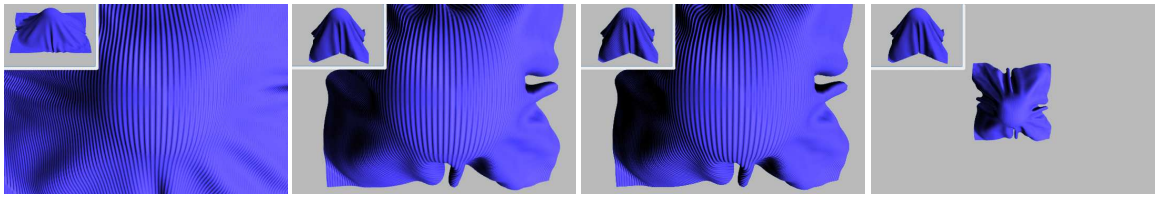
where $\boldsymbol{\omega}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)$ depends on the BRDF as usual (such as incident direction $\boldsymbol{\omega} = \boldsymbol{\omega}_i$ for Lambertian or halfway-vector $\boldsymbol{\omega} = \boldsymbol{\omega}_h$ for specular), and l^* is the maximum l used in the shader (accurate results generally require $l^* \approx \sqrt{4s}$ where s is the Blinn-Phong exponent). For shading, assume a single point light source for now. At each surface location, we know the incident and outgoing directions, so it is easy to find the half-vector $\boldsymbol{\omega}_h$ or other parameterization $\boldsymbol{\omega}$, and then use the BRDF formula above for rendering.[7]

We implement equation 7.12 in a pixel shader using GLSL (see our website for example code). The spherical harmonics Y_{lm} are stored in floating point textures, as are the MIP-mapped NDF coefficients $\gamma_{lm}(q)$. Real-time frame rates are achieved comfortably for up to 64 spherical harmonic terms ($l^* \leq 7$, corresponding to a Blinn-Phong exponent $s \leq 12$ or a Torrance-Sparrow surface roughness $\sigma \geq 0.2$).

7.4.2 RESULTS

Lambertian Reflection: In the Lambertian case, using only nine spherical harmonic coefficients ($l \leq 2$) suffices [Ramamoorthi and Hanrahan, 2001]. An example is shown in Figure 7.3. This figure also shows the generality of our method in terms of the footprint

[7] Our spherical harmonic algorithm does not explicitly address color textures; a simple approximation would be to MIP-map them separately, and then modulate the scalar result in equation 7.12. A more correct approach to filtering material properties is discussed for our vMF method in Section 7.6.4.



(a) Our method, frame 1 (b) Our method, frame 2 (c) "Ground truth", frame 2 (d) Our method, zoomed out

Figure 7.4: **Temporal coherence.** Stills from a sequence of cloth draping over a sphere, with closeups indicating correct normal filtering using our spherical harmonic algorithm. Note the smooth transition from the center (almost no filtering) to the corners (fully filtered) in (b)—compare also with ground truth in (c). (d) is a zoomed out view that also filters correctly. We use a blue fabric material from the Matusik database as the BRDF.

for texel q, by using GPU-based anisotropic filtering, instead of MIP-mapping. Note that we preserve accuracy in far away regions of the plane, while naïve averaging of the normal produces a nearly flat surface that is much darker than the actual (as illustrated in Figure 7.2e).

Low-Frequency Specularities and Measured Reflectance: Our framework also accommodates specular materials with BRDF $f(\boldsymbol{\omega}_h \cdot \boldsymbol{n})$. The BRDF can also be changed at runtime, since it is entirely independent of the NDF. We have factored all of the materials in the database of [Matusik *et al.*, 2003], using the $f(\theta_h)g(\theta_d)$ factorization in [Lawrence *et al.*, 2006]. Figure 7.5 shows two examples of different materials, which we can switch between at runtime.

Figure 7.4 shows closeup views from an animation sequence of cloth draping over a sphere, using the blue fabric material from the Matusik database. Note the accuracy of our method (compare (b) with the supersampled "ground truth" in (c)). Also note the smooth transition between close (unfiltered) and distant (fully filtered) regions in (a) and (b), as well as the filtered zoomed out view in (d).

Discussion and Limitations Our spherical harmonic method is a practical approach for low-frequency materials. Unlike previous techniques, all operations are linear—no nonlinear fitting is required, and we can handle arbitrary lobe shapes and functions $f(\boldsymbol{\omega}_h \cdot$

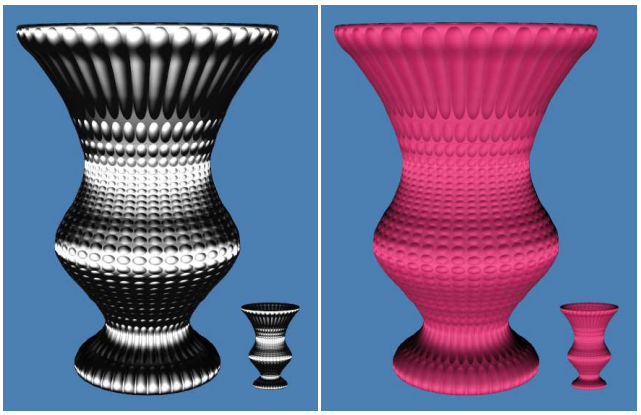


Figure 7.5: **Rendering with acquired BRDFs.** Our spherical harmonic algorithm for normal mapping, with two of the materials in the Matusik database—we can support general measured BRDFs and change reflectance or material in real time. Notice also the correct filtering of the zoomed out view, shown at the bottom right.

$\boldsymbol{n}$). Moreover, the BRDF is decoupled from the NDF, enabling simultaneous changes of BRDF, lighting and viewpoint.

As with all low-frequency approaches, our spherical harmonic method requires many terms for high-frequency specularities (a Blinn-Phong exponent of $s = 50$ needs about 200 coefficients). The following sections provide more practical solutions in these cases.

7.5 SPHERICALLY SYMMETRIC DISTRIBUTIONS

Spherical harmonics are a suitable basis for representing low-frequency functions, but are impractical for higher-frequency functions due to the large number of coefficients required. For higher-frequency NDFs, then, we will instead use radially symmetric basis functions, which are one-dimensional and therefore much more compactly represented. By performing an offline optimization, we approximate the NDF at each texel as the sum of a small number of such lobes.

7.5.1 BASIC THEORETICAL FRAMEWORK FOR USING SRBFS

Consider a single basis function $\gamma(\boldsymbol{n} \cdot \boldsymbol{\mu})$ for the NDF, symmetric about some central direction $\boldsymbol{\mu}$. For now, γ is a general *spherical radial basis function* (SRBF). Equation 7.10 now becomes

$$\rho^{\text{eff}}(\boldsymbol{\omega} \cdot \boldsymbol{\mu}; \gamma(\cdot)) = \int_{S^2} f(\boldsymbol{\omega} \cdot \boldsymbol{n}) \gamma(\boldsymbol{n} \cdot \boldsymbol{\mu}) \, d\boldsymbol{n}.$$

It can be shown (for example, see [Tsai and Shih, 2006]) that ρ^{eff} is itself radially symmetric about $\boldsymbol{\mu}$ (hence the form $\rho^{\text{eff}}(\boldsymbol{\omega} \cdot \boldsymbol{\mu})$ above), and its spherical harmonic coefficients are given by

$$\boxed{\rho_l^{\text{eff}} = \hat{\rho}_l \gamma_l.} \tag{7.13}$$

Compared to equation 7.11, this is a simpler 1D convolution, since all functions are radially symmetric and therefore one-dimensional. To represent general functions, we can use a small number of representative lobes $\gamma_{l,j}$. Note that the calculation of the lobe directions is generally a nonlinear process; our particular implementation is given in Section 7.6.

For rendering, we need to expand the effective BRDF in spherical harmonics, analogously to equation 7.12, but now using only the $m = 0$ terms. Considering the summation of J lobes, we obtain

$$\rho^{\text{eff}}(\boldsymbol{\omega}, q) = \sum_{j=1}^{J} \sum_{l=0}^{\infty} \hat{\rho}_l \gamma_{l,j}(q) Y_{l0}(\boldsymbol{\omega} \cdot \boldsymbol{\mu}_j), \tag{7.14}$$

where we again make clear that the NDF $\gamma_{l,j}$ is a function of the texel q. This equation can be used directly for shading once we find $\boldsymbol{\omega}$ for the light source and view direction.

7.5.2 DISCUSSION: UNIFYING FRAMEWORK AND MULTISCALE

Our theoretical framework in Section 7.5.1 unifies many normal filtering algorithms. Previous lobe- or peak-fitting methods Section 7.1 can be seen as special cases. By developing a general convolution framework, we show how to separate the NDF from the BRDF. Since we properly account for general BRDFs $\hat{\rho}_l$, we can even change BRDFs on the fly.

Equation 7.13 has an interesting multi-scale interpretation, as depicted in Figure 7.6. At the finest scale (a), the geometry used is the original highest-resolution normal map.

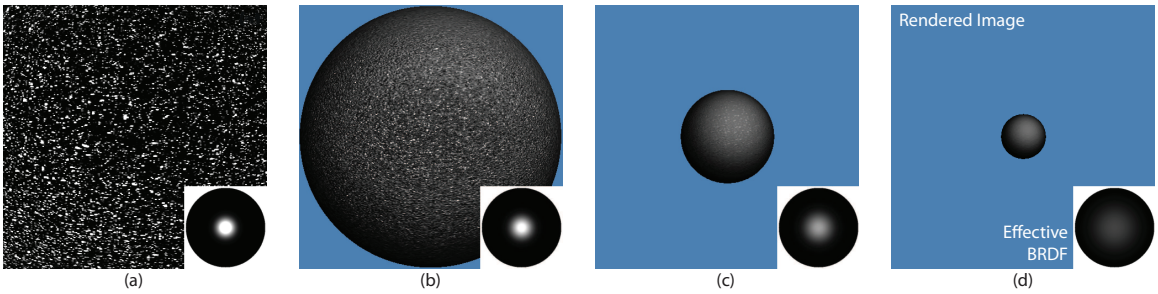


Figure 7.6: **Multiscale tradeoffs.** Illustration of filtering of the BRDF (rendered sphere) and NDF (inset). (a) shows a closeup of the sphere, where we see the individual facets and a sharp NDF/effective BRDF. In (b), we have zoomed out to where the geometry now appears smoother, although roughness is still clearly visible. The effective BRDF is now blurred, now incorporating finer-scale geometry. As we zoom further out in (c) and (d), the geometry appears even smoother, while the BRDF is further filtered.

Therefore, the NDF is a delta distribution at each texel, and the effective BRDF $\rho_l^{\text{eff}} = \hat{\rho}_l$. At coarser scales, the shading geometry used is effectively a filtered version of the fine-scale normal map, with the NDF becoming smoother from (b)-(d). The effective BRDF is now filtered by the smoothed NDF, essentially representing the complex fine-scale geometry as a blurring of the BRDF.

Also note the symmetry between the BRDF and NDF in equation 7.13. While the common fine-scale interpretation is for a delta function NDF and the original BRDF, we can also view it as a delta function BRDF and an NDF given by $\hat{\rho}_l$. These interpretations are consistent with most microfacet BRDF models, which start by assuming a mirror-like BRDF (delta function) and complex NDF (microfacet distribution), and derive a net glossy BRDF on a smooth macrosurface (delta function NDF).

7.5.3 CHOICE OF RADIAL BASIS FUNCTION

We now briefly discuss some possible approaches for approximating and representing our radial basis functions $\gamma(\boldsymbol{n} \cdot \boldsymbol{\mu})$. One possible method is to use zonal harmonics [Sloan *et al.*, 2005]; however, our high-frequency NDFs lead to large orders l, making fitting difficult and storage inefficient. An alternative is to use Gaussian RBFs, with parameters

chosen using expectation maximization (EM) [Dempster *et al.*, 1977]. In this case, we simply need to store 3 parameters per SRBF: the amplitude, width and central direction. Whereas Tan *et al.* [2005] pursued this approach using Euclidean or planar (and therefore distorted) RBFs, we consider NDFs represented on their natural spherical domain, which also enables us to derive a simple convolution formula.

Indeed, *spherical* Gaussian RBFs [Tsai and Shih, 2006] or Phong lobes [Fournier, 1992], are most appropriate. However, the nonlinear minimization required for fitting these models is inefficient, given that we need to do so at each texel. Instead, we use a spherical variant [Banerjee *et al.*, 2005] of EM, with the von Mises-Fisher[8] (vMF) distribution [Fisher, 1953]. Spherical EM and vMFs have previously been used in other areas such as computer vision [Hara *et al.*, 2005] for approximating Torrance-Sparrow BRDFs; here we introduce them for the first time in computer graphics, to represent NDFs.

7.6 VON MISES-FISHER MIXTURES

We now describe our algorithms for fitting the NDF, and rendering with mixtures of vMF lobes. The fitting is done using a technique known as spherical expectation maximization (EM) [Banerjee *et al.*, 2005]. EM is a common algorithm for fitting in statistics, that finds maximum-likelihood estimates of parameters [Dempster *et al.*, 1977]. It is an iterative method, with each iteration consisting of two steps known as the E-step and the M-step. We use EM as opposed to other fitting and minimization techniques because of its simplicity, efficiency, robustness, and ability to work with sparse data (the discrete normals in the NDF). We also show how to extend the basic spherical EM algorithm to handle color and different materials, create coherent lobes for hardware interpolation, and implement spherical harmonic convolution for rendering. Note that while the theoretical development of this section is somewhat complicated, the actual implementation is quite simple, and full pseudocode is provided in Algorithms 7.1 and 7.2.

[8] For the unit 3D sphere, this function is also known as the Fisher distribution. We use the more general term von Mises-Fisher distribution, that applies to n-dimensional hyperspheres.

7.6.1 FITTING NDFS WITH MIXTURES OF VMFS

vMF distributions were introduced in statistics to model Gaussian-like distributions on the unit sphere (or hypersphere). An advantage of vMFs is that they are well suited to a spherical expectation maximization algorithm to estimate their parameters. They are characterized by two parameters $\theta = \{\kappa, \boldsymbol{\mu}\}$ corresponding to the inverse width κ and central direction $\boldsymbol{\mu}$. vMFs are normalized to integrate to 1, as required by a probability distribution, and are given by

$$\gamma(\boldsymbol{n} \cdot \boldsymbol{\mu}; \theta) = \frac{\kappa}{4\pi \sinh(\kappa)} e^{\kappa(\boldsymbol{n} \cdot \boldsymbol{\mu})}. \tag{7.15}$$

A *mixture of vMFs* (movMF) is defined as an affine combination of vMF lobes θ_j, with amplitude α_j, where $\sum_j \alpha_j = 1$,

$$\gamma(\boldsymbol{n}; \Theta) = \sum_{j=1}^{J} \alpha_j \gamma_j(\boldsymbol{n} \cdot \boldsymbol{\mu}_j; \theta_j).$$

Here, $\theta_j = \{\kappa_j, \mu_j\}$ characterizes a single vMF lobe, and Θ stores the parameters $\{\alpha_j, \theta_j\}_{j=1}^{J}$ of all J vMFs in the movMF.

We use spherical EM (Algorithm 7.1) to fit a movMF to the normals covered at each texel in the MIP-map. Line 5 of Algorithm 7.1 shows the E-step. For all normals $\boldsymbol{n}_i$ in a given texel, we compute the expected likelihood $\langle z_{ij} \rangle$ that $\boldsymbol{n}_i$ corresponds to lobe j. Lines 9-14 execute the M-step, which computes maximum likelihood estimates of the parameters. In practice, we seldom need more than 10 iterations, so the full EM algorithm for a 512×512 normal map converges in under 2 minutes. Note that this is an offline computation that needs to be done only once per normal map—unlike most previous work, it is also independent of the BRDF (and lighting).

Note the use of auxiliary variable $\boldsymbol{r}_j$ in line 11, which represents $\langle \mathbf{x}_j \rangle / \alpha_j$, where $\langle \mathbf{x}_j \rangle$ is the expected value of a random vector generated according to the scaled vMF distribution $\gamma(\mathbf{x}; \theta_j)$. The central normal $\boldsymbol{\mu}_j$ and the inverse width κ_j are related to $\boldsymbol{r}_j$ by

$$\begin{aligned} \boldsymbol{r} &= A(\kappa)\boldsymbol{\mu}, \\ \text{where } A(\kappa) &= \coth(\kappa) - \frac{1}{\kappa}. \end{aligned} \tag{7.16}$$

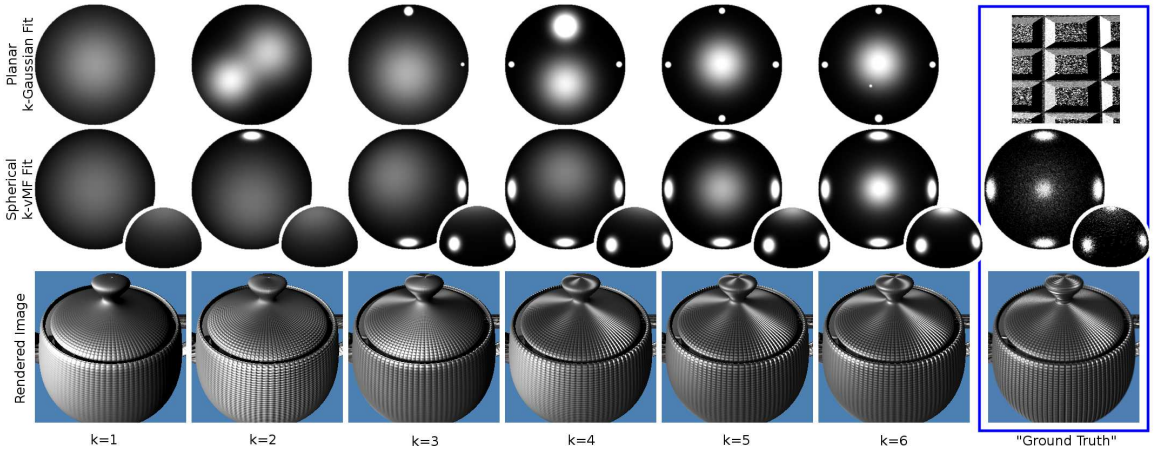


Figure 7.7: **vMF lobe fitting.** We fit the NDF using increasing numbers of lobes, at a representative MIP-map texel. The middle row displays our fitting results; with 3-4 lobes, we already get excellent agreement in the rendered image. Each vMF lobe is symmetric about some central direction, and is fit on the natural spherical domain (shown in both a top and side view, middle row). By contrast, a Gaussian EM fit on a planar projection of the hemisphere (top row), must remain symmetric in the distorted planar space, and has considerable errors at the boundaries of the hemisphere. Because no explicit convolution formula exists in the planar case, we only show renderings with our method (bottom row), which accurately match a reference with a few vMF lobes.

The direction $\boldsymbol{\mu}$ is found simply by normalizing $\boldsymbol{r}$ (line 13), while κ is given by $A^{-1}(\|\boldsymbol{r}\|)$; since no closed-form expression exists for A^{-1}, we use the approximation in [Banerjee *et al.*, 2005] (line 12).

Since EM is an iterative method, good initialization is important. For normal map filtering, we can proceed from the finest texels to coarser levels. At the finest level, we have only a single normal at each texel, so we need only a single lobe and directly set $\alpha = 1$, $\boldsymbol{\mu} = \boldsymbol{n}$, and κ to a large initial value. At coarser levels, a good initialization is to choose the furthest-apart J lobes from among the $4J$ $\boldsymbol{\mu}$'s in the four finer-level texels; for this we use Hochbaum-Shmoys clustering [Hochbaum and Shmoys, 1985]. Note that the actual fitting uses all normals covered by a given texel in the MIP-map.

The accuracy of our method is shown in Figure 7.7, where we see that about four lobes suffices in most cases, with excellent agreement with six lobes. We also compare with the

Algorithm 7.1 The Spherical EM algorithm. Inputs are normals $\boldsymbol{n}_i$ in a texel. Outputs are movMF parameters α, κ and μ for each lobe j.

repeat
  {The E-step}
  for all samples $\boldsymbol{n}_i$ **do**
    for $j = 1$ to J **do**
      $\langle z_{ij} \rangle \leftarrow \frac{\gamma_j(\boldsymbol{n}_i;\theta_j)}{\sum_{k=1}^{J} \gamma_k(\boldsymbol{n}_i;\theta_k)}$ {Expected likelihood of $\boldsymbol{n}_i$ in lobe j}
    end for
  end for
  {The M-step}
  for $j = 1$ to J **do**
    $\alpha_j \leftarrow \frac{\sum_{i=1}^{N} \langle z_{ij} \rangle}{N}$
    $\boldsymbol{r}_j \leftarrow \frac{\sum_{i=1}^{N} \langle z_{ij} \rangle \boldsymbol{n}_i}{\sum_{i=1}^{N} \langle z_{ij} \rangle}$ {Auxiliary variable for $\kappa, \boldsymbol{\mu}$ in equation 7.16}
    $\kappa_j \leftarrow \frac{3\|\boldsymbol{r}_j\| - \|\boldsymbol{r}_j\|^3}{1 - \|\boldsymbol{r}_j\|^2}$
    $\boldsymbol{\mu}_j \leftarrow \text{normalize}(\boldsymbol{r}_j)$
  end for
until convergence

Gaussian EM fits of Tan *et al.* [2005]. They work on a projection of the hemisphere onto the plane, and use standard Euclidean (rather than spherical) EM. Because this planar projection introduces distortions, they have a significant loss of accuracy near the boundaries (top row). Our method (middle row) works on the natural spherical domain (hence the side view shown), and is able to fit undistorted lobes anywhere on the sphere. Also note that, in contrast to previous methods, our form permits an explicit convolution formula and thus can be combined with any BRDF to produce accurate renderings (bottom row).

7.6.2 SPHERICAL HARMONIC COEFFICIENTS FOR RENDERING

For rendering, we will need the spherical harmonic coefficients γ_l of a normalized vMF lobe. To the best of our knowledge, these coefficients are not found in the literature, so we derive them here based on reasonable approximations. First, for large κ, we can assume that $\sinh(\kappa) \approx e^{\kappa}/2$. In practice, this approximation is accurate as long as $\kappa > 2$, which is

almost always the case. Hence, the vMF in equation 7.15 becomes

$$\gamma(\boldsymbol{n}\cdot\boldsymbol{\mu};\theta) \approx \frac{\kappa}{2\pi}e^{-\kappa(1-\boldsymbol{n}\cdot\boldsymbol{\mu})}.$$

Let β be the angle between $\boldsymbol{n}$ and $\boldsymbol{\mu}$. Then, $1-\boldsymbol{n}\cdot\boldsymbol{\mu} = 1-\cos\beta$. For moderate κ, β must be small for the exponential to be nonzero. In these cases, $1-\cos\beta \approx \beta^2/2$, and we get a Gaussian form,

$$\gamma(\boldsymbol{n}\cdot\boldsymbol{\mu};\theta) \approx \frac{\kappa}{2\pi}e^{-\frac{\kappa}{2}\beta^2}. \tag{7.17}$$

Spherical harmonic coefficients of a Torrance-Sparrow model of similar form have previously been studied [Ramamoorthi and Hanrahan, 2001]. For notational simplicity, let $\Lambda_l = \sqrt{\frac{4\pi}{2l+1}}$. Then,

$$\gamma = \frac{e^{-\beta^2/(4\sigma^2)}}{4\pi\sigma^2} \quad \Rightarrow \quad \Lambda_l\gamma_l = e^{-(\sigma l)^2}. \tag{7.18}$$

Comparing with equation 7.17, we obtain $\sigma^2 = \frac{1}{2\kappa}$ and

$$\Lambda_l\gamma_l = e^{-\sigma^2 l^2} = e^{-\frac{l^2}{2\kappa}}. \tag{7.19}$$

This formula provides us the desired spherical harmonic coefficients γ_l for a vMF lobe, in terms of the inverse width κ.

Having obtained γ_l, we are now ready to proceed to rendering. Since each vMF lobe is treated independently, and the constants α_j and BRDF coefficients can be multiplied separately, we focus on convolving the normalized BRDF $\hat{\rho}_l$ with a single normalized vMF lobe γ_l. It is possible to directly use equation 7.19 for the vMF coefficients and equation 7.14 for rendering with general BRDFs. However, a much simpler method is available for the important special forms of Blinn-Phong and Torrance-Sparrow like BRDFs. First, consider a normalized Blinn-Phong model of the form,

$$\rho = f(\boldsymbol{\omega}_h\cdot\boldsymbol{n}) = \frac{s+1}{2\pi}(\boldsymbol{\omega}_h\cdot\boldsymbol{n})^s,$$

where s is the specular exponent or shininess. It can be shown [Ramamoorthi and Hanrahan, 2001] that the spherical harmonic coefficients are $\hat{\rho}_l \approx e^{-l^2/2s}$. Therefore, the result

after convolution with the vMF is still approximately a Blinn-Phong shape:

$$\begin{aligned} \Lambda_l \rho_l^{\text{eff}} &= \hat{\rho}_l \Lambda_l \gamma_l = e^{-l^2/2s} e^{-l^2/2\kappa} = e^{-l^2/2s'}, \\ s' &= \frac{\kappa s}{\kappa + s} \\ \implies \rho^{\text{eff}}(\boldsymbol{\omega}_h \cdot \boldsymbol{\mu}) &= \frac{s'+1}{2\pi} (\boldsymbol{\omega}_h \cdot \boldsymbol{\mu})^{s'}. \end{aligned} \tag{7.20}$$

For a Torrance-Sparrow like BRDF of the form of equation 7.18, we obtain a similar form for ρ^{eff}, only with a new surface roughness σ' in the Torrance-Sparrow model, given by

$$\sigma' = \sqrt{\sigma^2 + (2\kappa)^{-1}}. \tag{7.21}$$

Equations 7.20 and 7.21 can easily be implemented in a GPU shader for rendering (lines 12-13 in Algorithm 7.2 implement equation 7.20; the full Algorithm 7.2 is explained at the end of Section 7.6.4). The simplicity of these formulae allows us to change BRDF parameters on the fly, and also to consider very high-frequency BRDFs.

7.6.3 COMPLEX LIGHTING

Our vMF-based normal map filtering technique can also be extended to complex environment map lighting.[9] Equation 7.2, rephrased below, is a convolution (mathematically similar to equation 7.10), that becomes a simple dot product in spherical harmonics,

$$B(\boldsymbol{\mu}) = \int_{S^2} L(\boldsymbol{\omega}_i) \rho^{\text{eff}}(\boldsymbol{\omega} \cdot \boldsymbol{\mu}) \, d\boldsymbol{\omega}_i, \tag{7.22}$$

where the effective BRDF ρ^{eff} is the convolution of the vMF lobe with the BRDF, and $\boldsymbol{\mu}$ is the central direction of the vMF lobe (effective "normal") as usual. For the diffuse or Lambertian component of the BRDF $\boldsymbol{\omega}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = \boldsymbol{\omega}_i$, and the spherical harmonic coefficients can simply be multiplied according to the convolution formula, $B_{lm} = \Lambda_l \rho_l^{\text{eff}} L_{lm}$, so that

$$B = \sum_{l=0}^{l^*} \sum_{m=-l}^{l} \Lambda_l \rho_l^{\text{eff}} L_{lm} Y_{lm}(\boldsymbol{\mu}). \tag{7.23}$$

[9] The direct spherical harmonic method in Section 7.4 is more difficult to apply, since general spherical harmonics cannot be rotated as easily as radially symmetric functions between local and global frames.

However, the specular component of the BRDF is expressed in terms of $\boldsymbol{\omega}(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = \boldsymbol{\omega}_h$, and we need to change the variable of integration in equation 7.22 to $\boldsymbol{\omega}_h$ (which leads to a factor $4(\boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_h)$),

$$\begin{aligned} B(\boldsymbol{\mu}) &= \int_{S^2} \left[L(\boldsymbol{\omega}_i(\boldsymbol{\omega}_h, \boldsymbol{\omega}_o)) \cdot 4(\boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_h)\right] \rho^{\text{eff}}(\boldsymbol{\omega}_h \cdot \boldsymbol{\mu})\, d\boldsymbol{\omega}_h \\ &= \int_{S^2} L'(\boldsymbol{\omega}_h) \rho^{\text{eff}}(\boldsymbol{\omega}_h \cdot \boldsymbol{\mu})\, d\boldsymbol{\omega}_h. \end{aligned}$$

Thus, we simply need to consider a new reparameterized lighting $L'(\boldsymbol{\omega}_h) = L(\boldsymbol{\omega}_i(\boldsymbol{\omega}_h, \boldsymbol{\omega}_o)) \cdot 4(\boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_h)$. As the half angle depends on both viewing and lighting angles ($\boldsymbol{\omega}_o$ and $\boldsymbol{\omega}_i$), the above integration implicitly limits us to a fixed view with respect to the lighting. To interactively rotate the lighting, we precompute a sparse set (typically, about 16×16) of rotated lighting coefficients and interpolate the shading.

Finally, in analogy with equation 7.23,

$$B = \sum_{l=0}^{l^*} \sum_{m=-l}^{l} \Lambda_l \rho_l^{\text{eff}} L'_{lm} Y_{lm}(\boldsymbol{\mu}). \tag{7.24}$$

7.6.4 EXTENSIONS

We now consider two practically important extensions: the augmentation of the vMF lobe model to support colors (Section 7.6.4.1), and alignment of neighboring lobes for accurate linear interpolation (Section 7.6.4.2). These extensions will result in changes to the basic EM algorithm (Section 7.6.4.3).

7.6.4.1 DIFFERENT MATERIALS/COLORS:

It is often the case that one would like to associate additional spatially varying properties (such as colors, material blending weights, etc.) to a normal map. For example, the normal map in Figure 7.2 contains regions of different colors. We represent these properties in a feature vector $\boldsymbol{y}_i$ associated with each normal $\boldsymbol{n}_i$, and extend the EM algorithm accordingly.

For each vMF lobe, we would now like to find a $\boldsymbol{y}_j$ that best describes the $\boldsymbol{y}_i$ of all its underlying texels. In Section 7.6.4.3, we augment the EM likelihood function with an

additional term whose maximization yields an extra line in the M-step,

$$\boldsymbol{y}_j \leftarrow \frac{\sum_{i=1}^{N} \langle z_{ij} \rangle \boldsymbol{y}_i}{\sum_{i=1}^{N} \langle z_{ij} \rangle} \tag{7.25}$$

Note that, since $\boldsymbol{y}_j$ does not affect the E-step, the preceding can be run as a postprocess to the vanilla EM algorithm.

This extension enables correct filtering of spatially-varying materials (as in Figure 7.2). Note however that only linear blending of basis BRDFs (and not for example, freely varying specular exponents) is allowed. Moreover, the result is a "best-fit" approximation, since normals and colors are assumed decorrelated.

7.6.4.2 COHERENT LOBES FOR HARDWARE INTERPOLATION:

In our case, accurate rendering involves shading the 8 neighboring MIP-map texels (using the BRDF and respective movMFs), and then trilinearly interpolating them with appropriate weights. Greater efficiency (usually a $2\times$ to $4\times$ speedup) is obtained if we instead follow the classic hardware approach of first trilinearly interpolating the parameters Θ of the movMFs. We can then simply run our GPU pixel shader once on the interpolated parameters $\tilde{\Theta}$. For accurate interpolation, this requires us to construct the movMFs in the MIP-map such that lobe j of each texel be similarly aligned to the jth lobe stored at each neighboring texel.

For alignment, we introduce a new term in our EM likelihood function, and maximize (details are provided in Section 7.6.4.3). The final result replaces line 13 in the M-step of Algorithm 7.1 with

$$\boldsymbol{\mu}_j \leftarrow \text{normalize}\left(\boldsymbol{r}_j + C \sum_{k=1}^{K} \alpha_{jk} \boldsymbol{\mu}_{jk}\right). \tag{7.26}$$

C is a parameter that controls the strength of alignment (intuitively, it seeks to move $\boldsymbol{\mu}_j$ closer to the central directions $\boldsymbol{\mu}_{jk}$ of the K neighbors, favoring neighbors with larger amplitudes α_{jk}.).

We build our aligned movMFs starting at the topmost (that is, most filtered) MIP-map level and proceed downward, following scanline ordering within each individual level. In the interest of performance, we use only previously visited texels as neighbors.

We next consider trilinear interpolation of the variables. Unfortunately, the customary vMF parameters $\{\kappa, \boldsymbol{\mu}\}$ control non-linear aspects of the vMF lobe and therefore cannot be linearly interpolated. To solve this problem, we recall from Section 7.6.1 that $\boldsymbol{\mu}$ and κ can be inferred from the scaled Euclidean mean $\boldsymbol{r} = \langle \mathbf{x} \rangle / \alpha$ of a given vMF distribution. By linearity of expectation, we can interpolate $\alpha \boldsymbol{r} = \langle \mathbf{x} \rangle$ linearly, as well as the amplitude α, giving

$$\tilde{\alpha}_j = T(\alpha_j) \qquad\qquad \tilde{\boldsymbol{r}}_j = T(\alpha_j \boldsymbol{r}_j)/T(\alpha_j),$$

where $T(\cdot)$ denotes trilinear hardware interpolation. Finally, $\tilde{\kappa}_j$ and $\tilde{\boldsymbol{\mu}}_j$ can easily be found in-shader (lines 9 and 10 of Algorithm 7.2).

Algorithm 7.2 shows pseudocode for our GLSL fragment shader. Lines 5-10 look up α and $\alpha \boldsymbol{r}$, and then compute κ and μ. For implementation, we store the jth lobe of each movMF in a standard RGBA MIP-map (vMFTexture in Algorithm 7.2) using one channel for α and one channel each for the three components of $\alpha \boldsymbol{r}$. Normalized color/material properties $\alpha \boldsymbol{y}$ are stored in corresponding textures (colorTexture in line 6 of Algorithm 7.2). Line 5 reads the parameters θ for a single vMF lobe as an RGBA value. Lines 12-13 compute the specular shading (assuming a Blinn-Phong model with exponent s) using equation 7.20. The Torrance-Sparrow model can be handled similarly, using equation 7.21. Line 14 computes the final shading contribution by including the color parameters $\boldsymbol{y}$, and scaling by the lobe amplitude α, specular coefficient K_s, and the cosine of the incident angle, while adding the Lambertian component K_d. Note that this shader can be used equally with aligned or unaligned vMF lobes; the only difference is whether we manually compute and combine results from all 8 neighboring texels (unaligned) or use hardware interpolate to first obtain lobe parameters (aligned).

7.6.4.3 AUGMENTED FITTING ALGORITHM

The additions of the preceding subsections result in a change to the likelihood function for spherical EM. The net likelihood function is a product of 3 terms,

$$P(X, Z|\Theta) P(Y, Z|\Theta) P(\Theta|N(\Theta)),$$

Algorithm 7.2 Pseudocode for the vMF GLSL fragment shader.

{Setup: calculate half angle $\boldsymbol{\omega}_h$ and incident angle $\boldsymbol{\omega}_i$}
$\rho \leftarrow 0$
for $j = 1$ to J **do** {Add up contributions for all J lobes}
  {Look up vMF parameters stored in 2D texture map}
    $\theta \leftarrow$ texture2D(vMFTexture$[j], s, t$)
    $a\boldsymbol{y} \leftarrow$ texture2D(colorTexture$[j], s, t$)
    $a \leftarrow \theta.x$
    $\boldsymbol{r} \leftarrow \frac{\theta.yzw}{a}$ {$\theta.yzw$ stores $a\boldsymbol{r}$}
    $\kappa \leftarrow \frac{3\|\boldsymbol{r}\| - \|\boldsymbol{r}\|^3}{1-\|\boldsymbol{r}\|^2}$
    $\boldsymbol{\mu} \leftarrow$ normalize$(\boldsymbol{r})$
  {Calculate shading per equation 7.20}
    $s' \leftarrow \frac{\kappa s}{\kappa + s}$ {s is Blinn-Phong exponent}
    $B_s \leftarrow \frac{s'+1}{2\pi}(\boldsymbol{\omega}_h \cdot \boldsymbol{\mu})^{s'}$ {Equation 7.20}
  $\rho \leftarrow \rho + a\boldsymbol{y}(K_s B_s + K_d)(\boldsymbol{\omega}_i \cdot \boldsymbol{\mu})$
end for
gl_FragColor $\leftarrow L \times \rho$ {L is light intensity}

where X are the samples (in this case input normals), Y are the colors/materials, Z are the hidden variables (in this case which vMF lobe a sample X is drawn from), Θ are parameters for all vMF lobes and $N(\Theta)$ are parameters for neighbors. The first factor corresponds to standard spherical EM, the second factor corresponds to the colors/materials Y,

$$P(Y, Z|\Theta) = \prod_{i=1}^{N} e^{-\|\boldsymbol{y}_{z_i} - \boldsymbol{y}_i\|^2},$$

and the final factor to coherent lobes for interpolation,

$$P(N(\Theta)|\Theta) = \prod_{j=1}^{J}\prod_{k=1}^{K} e^{C'\alpha_{jk}(\boldsymbol{\mu}_j \cdot \boldsymbol{\mu}_{jk})}.$$

We use C' above as a constant weighting factor (it will be related to the weight C used in the main text as discussed below).

In EM, we seek to maximize the log likelihood

$$\ln\left[P(X, Z|\Theta)P(Y, Z|\Theta)P(\Theta|N(\Theta))\right] = \sum_{i=1}^{N} \ln \gamma(\boldsymbol{n}_i|\theta_{z_i}) + \sum_{i=1}^{N} -\|\boldsymbol{y}_i - \boldsymbol{y}_{z_i}\|^2 + \sum_{j=1}^{J}\sum_{k=1}^{K} C'\alpha_{jk}(\boldsymbol{\mu}_j \cdot \boldsymbol{\mu}_{jk}),$$

which, considering all J lobes and hidden variables $\langle z_{ij} \rangle$, becomes

$$\sum_{j=1}^{J} \left[\sum_{i=1}^{N} \ln \gamma(\boldsymbol{n}_i | \theta_j) \langle z_{ij} \rangle + \sum_{i=1}^{N} -\| \boldsymbol{y}_i - \boldsymbol{y}_{z_i} \|^2 \langle z_{ij} \rangle + \sum_{k=1}^{K} C' \alpha_{jk} (\boldsymbol{\mu}_j \cdot \boldsymbol{\mu}_{jk}) \right] .$$

Maximizing with respect to $\boldsymbol{y}_j$, we directly obtain equation 7.25. The maximization with respect to $\boldsymbol{\mu}_j$ is more complex,

$$\boldsymbol{\mu}_j = \text{normalize} \left(\kappa_j \sum_{i=1}^{N} \boldsymbol{n}_i \langle z_{ij} \rangle + C' \sum_{k=1}^{K} \alpha_{jk} \boldsymbol{\mu}_{jk} \right) .$$

Finally, redefining $C = C'/\kappa_j$, we obtain equation 7.26.

7.6.5 RESULTS

Figure 7.2 shows the accuracy of our method, and makes comparisons to ground truth and alternative techniques. It also shows our ability to use different materials for different parts of the normal map.

Our formulation allows for general and even dynamically changing BRDFs. Figure 7.8 shows a complex scene, where the reflectance changes over time, decreasing in shininess. Although not shown, the lighting and view can also vary—the bottom row shows close-ups with different illumination. Note the correct filtering for dinosaurs in the background, and for further regions along the neck and body of the foreground dinosaur. Even where individual bumps are not visible, the overall change in appearance as the reflectance changes is clear. This complex scene has 14,898 triangles for the dinosaurs, 139,392 triangles for the terrain and 6 different textures and normal maps for the dinosaur skins. It renders at 75 frames per second at a resolution of 640x480 on an nVIDIA 8800 graphics card. In this example, we used six vMF lobes, with both diffuse and specular shading implemented as a simple fragment shader.

Finally, Figure 7.9 shows an image of an armadillo rendered under dynamic (rotating) environment map lighting. We were able to render this model (approximately 350,000 polygons) at interactive framerates, with up to 6 vMF lobes and $l^* = 8$ in Equation 7.24.

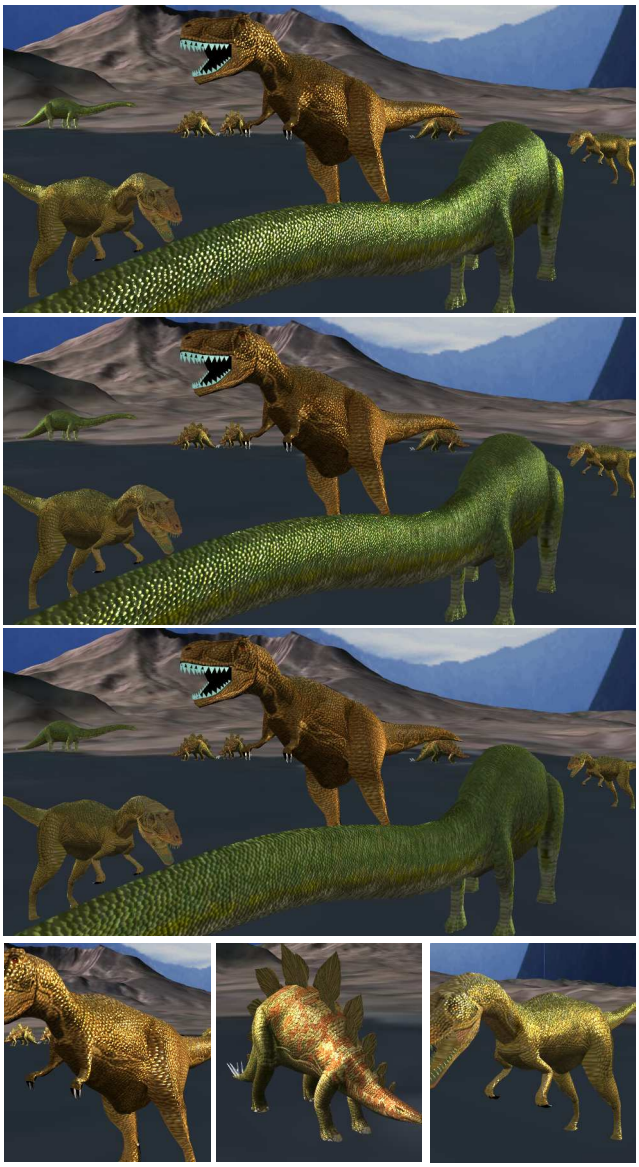


Figure 7.8: **Dynamically changing reflectance.** Our framework can handle complex scenes, allowing for general reflectance, which can even be changed at run-time. Here, the BRDF becomes less shiny over time. Note the correct filtering and overall changes in appearance for further regions of the foreground dinosaur, and those in the background. The bottom row shows closeups (when the material is shiny) with a different lighting condition. This example also shows that we can combine filtered normal maps with standard color texture mapping.

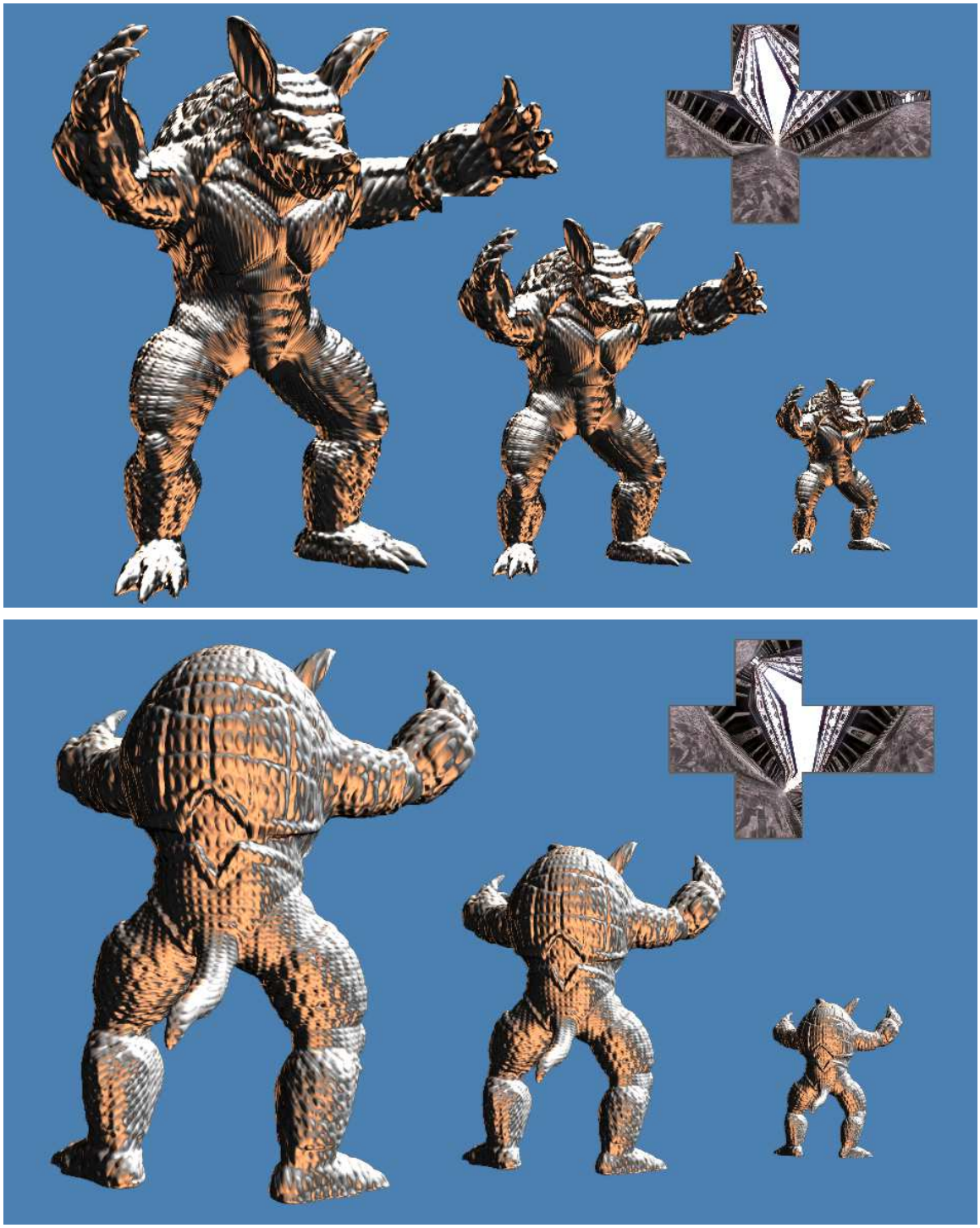


Figure 7.9: **Normal map filtering under complex lighting.** Armadillo model with 350,000 polygons rendered interactively with normal maps in dynamic environment lighting. We use 6 vMF lobes, and spherical harmonics up to order 8 for the specular component.

Part IV

CONCLUSIONS

CHAPTER 8

FUTURE DIRECTIONS

It will become necessary to develop multiscale analogues to many of the existing methods from the traditional, narrow-scale setting.

Incorporating geometry In developing the foundations of a multiscale approach, we have thus far operated chiefly in the planar 2D image domain. However, an important avenue for future research will be in applying these concepts towards synthesis directly on mesh geometry. Although there has been a good deal of research into surface texture synthesis [Turk, 2001; Wei and Levoy, 2001; Ying *et al.*, 2001; Zhang *et al.*, 2003; Magda and Kriegman, 2003; Lefebvre and Hoppe, 2006], these methods (much like their planar counterparts) are not particularly suited for the multiscale setting. In the area of texture editing, modern "final placement" tools allow painting directly on the mesh, hiding the details of parameterization, or avoiding parameterization altogether [Burley and Lacewell, 2008]. We hope to see the application of concepts from our multiscale editing framework into these tools.

This move to incorporate geometry is especially compatible with future directions in filtering as well. Several authors have spelled out a hierarchy of levels-of-detail, including: explicit 3D geometry, height displacements, normal/bump maps, and BRDF or reflectance [Kajiya, 1985; Becker and Max, 1993]. Our work has addressed filtering of normal maps and to some extent, the transition to a BRDF at far distances. However, a critical direction for future work is filtering of geometry or displacement maps, where

effects like local occlusions, shadowing, masking and interreflections become important. With more and more geometry processing being moved into the real-time domain (*e.g.*, geometry and tessellation shaders), we anticipate that it will become possible to develop a unified approach to filtering the entire hierarchy of visual detail.

Lastly, one might consider extensions of our methods into higher-dimensional domains such as solid [Kopf *et al.*, 2007a] or time-varying [Schödl *et al.*, 2000; Gu *et al.*, 2006] textures. Solid textures in particular have proven to be useful in texturing complex objects where a surface parameterization is unavailable or otherwise undesirable.

General imagery Just as we are interested in larger and more complex textures, there has been an increasing interest in larger imagery in general [Kopf *et al.*, 2007b]. Although we have maintained a distinction in this thesis between "textural" and general, "structural" images, there are many cases where such a distinction is not entirely clear. Many–if not most–general images contain textural elements, and can therefore still benefit from judicious employment of ideas from texturing. Such applications include texture replacement [Tsin *et al.*, 2001; Liu *et al.*, 2004] and direct use of perspective photographs as exemplars [Eisenacher *et al.*, 2008].

Data-driven methods We have thus far worked to allow careful selection and control of exemplars for texturing. An alternative approach is to pre-assemble and analyze a large database of textures, from which a user can later select a subset for recombination [Matusik *et al.*, 2005]. Given the growing sea of available source imagery, this approach will become increasingly attractive if it can be adopted for larger image sizes. Imagine, for instance, a "semantic" paintbrush tool that fills in image regions with a desired texture drawn from a database of choices, all while avoiding issues such as the mid-frequency problem (Section 1).

CHAPTER 9

SUMMARY AND FINAL WORDS

In this thesis we have introduced the multiscale setting for textures, and have presented several representations and methods for working in this new domain.

A subtle but important consequence of our work is simply that it is possible at all to work directly with multiscale textures. This is not an immediately obvious observation, as evidenced by current industry practices. In motion pictures, for instance, it is common for textures to be generated "per-shot"; the same object may require completely different texture assets for close-up and wide-angle shots, due to the representational limits of traditional textures. We reject this pattern and offer an alternative view: the same texture should be usable at any scale, through a careful development of both representation and algorithm.

Indeed, our work has been in many ways driven by finding the correct multiscale representations. Our exemplar graph and accompanying synthesis and editing frameworks provide an intuitive method for generating multiscale textures. Likewise, our NDF representation and ensuing discretization choices form the key to accurate normal map filtering. We needed to discard the old representations—monolithic exemplars and surface normal vectors, respectively—before we could begin to form new methods.

Our algorithms have been developed throughout with an eye towards practicality and usability, with GPU implementations described and demonstrated for many of our methods. Furthermore, we have shown that it is possible to import many of the insights

from single-scale texture methods into the multiscale setting.

While we have dealt here with the most fundamental issues of synthesis, editing, and display, we view these ultimately as important first steps towards a more complete understanding of multiscale textures.

BIBLIOGRAPHY

[Artač *et al.*, 2002] Matej Artač, Matjaž Jogan, and Aleš Leonardis. Incremental PCA for on-line visual learning and recognition. In *Proceedings of the International Conference on Pattern Recognition*, pages 781–784, 2002. 38

[Ashikhmin, 2001] Michael Ashikhmin. Synthesizing natural textures. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 217–226, 2001. 24, 37

[Avidan and Shamir, 2007] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics*, 26(3):10, 2007. 37

[Banerjee *et al.*, 2005] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von Mises-Fisher distributions. *Journal of Machine Learning Research*, 6:1345–1382, 2005. 71, 73

[Bar-Joseph *et al.*, 2001] Ziv Bar-Joseph, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, 2001. 20

[Barnes *et al.*, 2009] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 28(3):3, 2009. 37, 39

[Barnes *et al.*, 2010] Connelly Barnes, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. The generalized PatchMatch correspondence algorithm. In *Proceedings of the European Conference on Computer Vision*, 2010. 39

[Basri and Jacobs, 2001] Ronen Basri and David Jacobs. Lambertian reflectance and linear subspaces. In *Proceedings of the International Conference on Computer Vision*, pages 383–390, 2001. 57, 63

[Becker and Max, 1993] Barry Becker and Nelson Max. Smooth transitions between bump rendering algorithms. In *Proceedings of ACM SIGGRAPH*, pages 183–190, 1993. 57, 85

[Blinn, 1978] James F. Blinn. Simulation of wrinkled surfaces. In *Proceedings of ACM SIGGRAPH*, pages 286–292, 1978. 53

[Burley and Lacewell, 2008] Brent Burley and Dylan Lacewell. Ptex: Per-face texture mapping for production rendering. *Computer Graphics Forum*, 26(4):1155–1164, 2008. 85

[Busto *et al.*, 2010] Pau Panareda Busto, Christian Eisenacher, Sylvain Lefebvre, and Marc Stamminger. Instant texture synthesis by numbers. *Vision, Modeling and Visualization*, pages 81–85, 2010. 37

[Charalampidis, 2006] Dimitrios Charalampidis. Texture synthesis: textons revisited. *IEEE Transactions on Image Processing*, 15(3):777–787, 2006. 10

[Cheng *et al.*, 2010] Ming-Ming Cheng, Fang-Lue Zhang, Niloy J. Mitra, Xiaolei Huang, and Shi-Min Hu. RepFinder: Finding approximately repeated scene elements for image editing. *ACM Transactions on Graphics*, 29(4):83:1–83:8, 2010. 37

[Cho *et al.*, 2010] Taeg Sang Cho, Shai Avidan, and William T. Freeman. The patch transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1489–1501, 2010. 37

[Claustres *et al.*, 2007] Luc Claustres, Loïc Barthe, and Mathias Paulin. Wavelet encoding of BRDFs for real-time rendering. In *Proceedings of Graphics Interface*, pages 169–176, 2007. 57

[Cohen *et al.*, 1998] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance preserving simplification. In *Proceedings of ACM SIGGRAPH*, pages 115–122, 1998. 57

[Cross and Jain, 1983] George R. Cross and Anil K. Jain. Markov random field texture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(1):25–39, 1983. 6

[Dagher and Nachar, 2006] Issam Dagher and Rabih Nachar. Face recognition using IPCA-ICA algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:966–1000, 2006. 38

[De Bonet, 1997] Jeremey S De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of ACM SIGGRAPH*, pages 361–368, 1997. 9, 20

[Dempster *et al.*, 1977] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977. 71

[Dischler *et al.*, 2002] Jean-Michel Dischler, Karl Maritaud, Bruno Lévy, and Djamchid Ghazanfarpour. Texture particles. *Computer Graphics Forum*, 21(3):401–410, 2002. 10

[Ebert *et al.*, 2003] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, San Francisco, CA, 2003. 15

[Efros and Freeman, 2001] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH*, pages 341–346, 2001. 8, 20

[Efros and Leung, 1999] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision*, pages 1033–1038, 1999. 7, 20

[Eisenacher *et al.*, 2008] Christian Eisenacher, Sylvain Lefebvre, and Marc Stamminger. Texture synthesis from photographs. *Computer Graphics Forum*, 27:419–428, 2008. 86

[Eisenacher *et al.*, 2010] Christian Eisenacher, Chuck Tappan, Brent Burley, Daniel Teece, and Arthur Shek. Example-based texture synthesis on Disney's Tangled. In *ACM SIGGRAPH Talks*, pages 32:1–32:1, 2010. 37

[Farbman *et al.*, 2009] Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski. Coordinates for instant image cloning. *ACM Transactions on Graphics*, 28:67:1–67:9, July 2009. 37

[Fisher, 1953] Ronald Fisher. Dispersion on a sphere. *Proceedings of the Royal Society of London, Series A*, 217:295–305, 1953. 71

[Fournier, 1992] Alain Fournier. Normal distribution functions and multiple surfaces. In *Proceedings of the Graphics Interface Workshop on Local Illumination*, pages 45–52, 1992. 56, 71

[Freeman *et al.*, 2001] William T. Freeman, Thouis R. Jones, and Egon C. Pasztor. Example-based super-resolution. Technical Report TR-2001-30, Mitsubishi Electric Research Laboratories, 2001. 30

[Gu *et al.*, 2006] Jinwei Gu, Chien-I Tu, Ravi Ramamoorthi, Peter Belhumeur, Wojciech Matusik, and Shree Nayar. Time-varying surface appearance: Acquisition, modeling and rendering. *ACM Transactions on Graphics*, 25(3):762–771, 2006. 37, 86

[Han and Hoppe, 2010] Charles Han and Hugues Hoppe. Optimizing continuity in multiscale imagery. *ACM Transactions on Graphics*, 29(5):171:1–171:9, 2010. 18

[Hara *et al.*, 2005] Kenji Hara, Ko Nishino, and Katsushi Ikeuchi. Multiple light sources and reflectance property estimation based on a mixture of spherical distributions. In *Proceedings of the International Conference on Computer Vision*, pages 1627–1634, 2005. 71

[Heckbert, 1989] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Technical Report UCB/CSD-89-516, University of California, Berkeley, 1989. 53

[Heeger and Bergen, 1995] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of ACM SIGGRAPH*, pages 229–238, 1995. 8, 20

[Hertzmann *et al.*, 2001] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of ACM SIGGRAPH*, pages 327–340, 2001. 30, 37

[Hochbaum and Shmoys, 1985] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 1985. 73

[Julesz, 1981] Béla Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290:91–97, 1981. 5

[Kajiya, 1985] James T. Kajiya. Anisotropic reflection models. In *Proceedings of ACM SIGGRAPH*, pages 15–21, 1985. 57, 85

[Kopf *et al.*, 2007a] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2D exemplars. *ACM Transactions on Graphics*, 26(3):2, 2007. 9, 86

[Kopf *et al.*, 2007b] Johannes Kopf, Matt Uyttendaele, Oliver Deussen, and Michael F. Cohen. Capturing and viewing gigapixel images. *ACM Transactions on Graphics*, 26, 2007. 86

[Kwatra *et al.*, 2003] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003. 8, 20

[Kwatra *et al.*, 2005] Vivek Kwatra, Irfan Essa, Aaron F. Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics*, 24(3):795–802, 2005. 8

[Lalonde and Fournier, 1997] Paul Lalonde and Alain Fournier. A wavelet representation of reflectance functions. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):329–336, 1997. 57

[Lawrence *et al.*, 2006] Jason Lawrence, Aner Ben-Artzi, Chris Decoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. Inverse shade trees

for non-parametric material representation and editing. *ACM Transactions on Graphics*, 25(3):735–745, 2006. 59, 67

[Lefebvre and Hoppe, 2005] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Transactions on Graphics*, 24(3):777–786, 2005. 8, 19, 20, 21, 22, 23, 24, 27, 37, 38, 49

[Lefebvre and Hoppe, 2006] Sylvain Lefebvre and Hugues Hoppe. Appearance-space texture synthesis. *ACM Transactions on Graphics*, 25(3):541–548, 2006. 10, 37, 85

[Leung and Malik, 2001] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using 3d textons. *International Journal of Computer Vision*, 43(1):29–44, 2001. 9

[Liang *et al.*, 2001] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Harry Shum. Real-time texture synthesis by patch-based sampling. Technical Report MSR-TR-2001-40, Microsoft Research, 2001. 20

[Liu *et al.*, 2004] Yanxi Liu, Wen-Chieh Lin, and James Hays. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics*, pages 368–376, 2004. 10, 86

[Magda and Kriegman, 2003] Sebastian Magda and David J. Kriegman. Fast texture synthesis on arbitrary meshes. In *Proceedings of the Eurographics Workshop on Rendering*, pages 82–89, 2003. 85

[Matusik *et al.*, 2003] Wojciech Matusik, Hanspeter Pfister, Matthew Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, 2003. 67

[Matusik *et al.*, 2005] Wojciech Matusik, Matthias Zwicker, and Frédo Durand. Texture design using a simplicial complex of morphable textures. *ACM Transactions on Graphics*, 24:787–794, 2005. 10, 20, 86

[Mohammed *et al.*, 2009] Umar Mohammed, Simon J.D. Prince, and Jan Kautz. Visiolization: generating novel facial images. *ACM Transactions on Graphics*, pages 57:1–57:8, 2009. 10

[Ng *et al.*, 2004] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics*, 23(3):475–485, 2004. 62

[Olano and Baker, 2010] Marc Olano and Dan Baker. LEAN mapping. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 181–188, 2010. 56

[Olano and North, 1997] Marc Olano and Michael North. Normal distribution mapping. Technical Report 97-041, UNC, 1997. 56

[Paget, 2004] Rupert Paget. Strong markov random field model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):408–413, 2004. 7

[Perlin, 1985] Ken Perlin. An image synthesizer. In *Proceedings of ACM SIGGRAPH*, pages 287–296, 1985. 15

[Popat and Picard, 1993] Kris Popat and Rosalind W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *SPIE Visual Communications and Image Processing*, pages 756–768, 1993. 7, 20

[Portilla and Simoncelli, 2000] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–70, 2000. 8, 9, 20

[Praun *et al.*, 2000] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of ACM SIGGRAPH*, pages 465–470, 2000. 20

[Ramamoorthi and Hanrahan, 2001] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of ACM SIGGRAPH*, pages 117–128, 2001. 57, 58, 63, 66, 75

[Risser *et al.*, 2010] Eric Risser, Charles Han, Rozenn Dahyot, and Eitan Grinspun. Synthesizing structured image hybrids. *ACM Transactions on Graphics*, 29(4):85:1–85:6, 2010. 10, 49

[Ritter *et al.*, 2006] Lincoln Ritter, Wilmot Li, Maneesh Agrawala, Brian Curless, and David Salesin. Painting with texture. In *Proceedings of the Eurographics Symposium on Rendering*, pages 371–376, 2006. 37

[Rosenberger *et al.*, 2009] Amir Rosenberger, Daniel Cohen-Or, and Dani Lischinski. Layered shape synthesis: automatic generation of control maps for non-stationary textures. *ACM Transactions on Graphics*, 28(5):107, 2009. 4

[Schilling, 1997] Andreas Schilling. Towards real-time photorealistic rendering: Challenges and solutions. In *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 7–15, 1997. 56

[Schödl *et al.*, 2000] Arno Schödl, Richard Szeliski, David Salesin, and Irfan Essa. Video textures. In *Proceedings of ACM SIGGRAPH*, pages 489–498, 2000. 86

[Sloan *et al.*, 2002] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics*, 21(3):527–536, 2002. 57

[Sloan *et al.*, 2005] Peter-Pike Sloan, Ben Luna, and John Snyder. Local, deformable precomputed radiance transfer. *ACM Transactions on Graphics*, 24(3):1216–1224, 2005. 57, 70

[Tan *et al.*, 2005] Ping Tan, Stephen Lin, Long Quan, Baining Guo, and Harry Shum. Multiresolution reflectance filtering. In *Proceedings of the Eurographics Symposium on Rendering*, pages 111–116, 2005. 56, 71, 74

[Toksvig, 2005] Michael Toksvig. Mipmapping normal maps. *Journal of Graphics Tools*, 10(3):65–71, 2005. 55, 56

[Tong *et al.*, 2002] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics*, 21(3):665–672, 2002. 24, 39

[Tonietto and Walter, 2002] Leandro Tonietto and Marcelo Walter. Towards local control for image-based texture synthesis. In *Proceedings of SIBGRAPI*, page 252, 2002. 20

[Tsai and Shih, 2006] Yu-Ting Tsai and Zen-Chung Shih. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Transactions on Graphics*, 25(3):967–976, 2006. 57, 69, 71

[Tsin *et al.*, 2001] Yanghai Tsin, Yanxi Liu, and Visvanathan Ramesh. Texture replacement in real images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 539–544, 2001. 86

[Turk, 2001] Greg Turk. Texture synthesis on surfaces. In *Proceedings of ACM SIGGRAPH*, pages 347–354, 2001. 85

[Wei and Levoy, 2000] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH*, pages 479–488, 2000. 7, 20

[Wei and Levoy, 2001] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of ACM SIGGRAPH*, pages 355–360, 2001. 85

[Wei and Levoy, 2002] Li-Yi Wei and Marc Levoy. Order-independent texture synthesis. Technical Report TR-2002-01, Stanford University Computer Science Department, 2002. 20

[Wei *et al.*, 2008] Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Inverse texture synthesis. *ACM Transactions on Graphics*, 27(3):52, 2008. 4, 18, 37

[Wei, 2002] Li-Yi Wei. *Texture synthesis by fixed neighborhood searching*. PhD thesis, Stanford University, Stanford, California, 2002. 20, 40

[Williams, 1983] Lance Williams. Pyramidal parametrics. In *Proceedings of ACM SIGGRAPH*, pages 1–11, 1983. 53

[Wu and Yu, 2004] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics*, 23:364–367, 2004. 8

[Ying *et al.*, 2001] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and shape synthesis on surfaces. In *Proceedings of the Eurographics Workshop on Rendering*, pages 301–312, 2001. 85

[Zalesny *et al.*, 2005] Alexey Zalesny, Vittorio Ferrari, Geert Caenen, and Luc J. Van Gool. Composite texture synthesis. *International Journal of Computer Vision*, 62(1-2):161–176, 2005. 20

[Zelinka and Garland, 2002] Steve Zelinka and Michael Garland. Towards real-time texture synthesis with the jump map. In *Proceedings of the Eurographics Workshop on Rendering*, pages 99–104, 2002. 24, 45

[Zhang *et al.*, 2003] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics*, 22:295–302, 2003. 4, 10, 20, 85

[Zhu and Mumford, 1998] Song-Chun Zhu and David Mumford. Filters, random fields and maximum entropy (FRAME)–towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998. 7, 8

[Zhu *et al.*, 2005] Song-Chun Zhu, Cheng en Guo, Yizhou Wang, and Zijian Xu. What are textons? *International Journal of Computer Vision*, 62(1-2):121–143, 2005. 9