Artificial Intelligence for Efficient Image-based View Synthesis

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften der Fakultät für Mathematik und Informatik der Universität des Saarlandes

Vorgelegt von
Thomas Leimkühler

Saarbrücken, 2019

Dean:	Prof. Dr. Sebastian Hack
Date:	June 24th, 2019
Chair:	Prof. Dr. Philipp Slusallek
Reviewers:	Prof. Dr. Hans-Peter Seidel
	Dr. Tobias Ritschel
	Prof. Dr. Hendrik Lensch
	Prof. Dr. George Drettakis
Academic Assistant:	Dr. Rhaleb Zayer

Abstract

Synthesizing novel views from image data is a widely investigated topic in both computer graphics and computer vision, and has many applications like stereo or multi-view rendering for virtual reality, light field reconstruction, and image post-processing. While image-based approaches have the advantage of reduced computational load compared to classical model-based rendering, efficiency is still a major concern. This thesis demonstrates how concepts and tools from artificial intelligence can be used to increase the efficiency of image-based view synthesis algorithms. In particular it is shown how machine learning can help to generate point patterns useful for a variety of computer graphics tasks, how path planning can guide image warping, how sparsity-enforcing optimization can lead to significant speedups in interactive distribution effect rendering, and how probabilistic inference can be used to perform real-time 2D-to-3D conversion.

Zusammenfassung

Die bildbasierte Synthese von neuen Ansichten ist Gegenstand intensiver Forschungsbemühungen sowohl in der Computergrafik als auch im maschinellen Sehen, mit Anwendungen wie stereo- und multiskopisches Rendering, virtuelle Realität, Lichtfeldrekonstruktion und nachträgliche Bildverarbeitung. Bildbasierte Ansätze weisen den Vorteil auf, dass sie im Vergleich zum klassischen modellbasierten Rendering weniger Rechenleistung erfordern. Effizienz ist jedoch weiterhin ein großes Problem. Diese Arbeit zeigt, wie Konzepte und Hilfsmittel aus dem Feld der künstlichen Intelligenz benutzt werden können, um die Effizienz der bildbasierten Synthese von neuen Ansichten zu steigern. Im Einzelnen wird gezeigt, wie maschinelles Lernen das Generieren von Punktmustern, die für viele Anwendungen in der Computergrafik nützlich sind, unterstützen kann und wie das Planen von Pfaden das Warpen von Bildern in effiziente Bahnen lenken kann. Weiterhin wird eine Optimierung zur Ausdünnung von Datenstrukturen entwickelt, die das interaktive Rendern von Verteilungseffekten signifikant beschleunigt. Abschließend wird gezeigt, wie probabilistische Inferenz zur 2D-zu-3D Konvertierung in Echtzeit benutzt werden kann.

Acknowledgments

This work would have been impossible without the support of many. First and foremost, I would like to sincerely thank Tobias Ritschel. During our numerous insightful discussions, his vision, advice, and creativity gradually turned me into a researcher. Karol Myszkowski is probably the wisest person I know. He provided valuable guidance during our joint projects and beyond. The computer graphics department in the Max-Planck-Institut für Informatik is a wonderfully inspiring environment, which Hans-Peter Seidel created and maintains with generosity and the desire to provide the best possible opportunities. I would like to thank Petr Kellnhofer and Gurprit Singh for sharing their expertise in our pleasant common endeavors. Learning from and exchanging ideas with Rhaleb Zayer, Anton Kaplanyan, Bernhard Reinert, Jozef Hladký, Tobias Bertel, and Okan Tursun was not only fun, but also opened my mind in many unexpected directions. A big round of thanks goes to all members of AG4 for enriching my academic and personal life. Finally, I am deeply grateful to my friends and family, who continually help me discover the most important novel views.

Contents

Chapter 1

Introduction

This thesis proposes several techniques that accelerate and extend the scope of image-based view synthesis algorithms. In this chapter we motivate our research (Sec. 1.1), summarize our main contributions, (Sec. 1.2), and give an outline of the whole thesis (Sec. 1.3).

1.1 Motivation

In 2014 the artists Denis Conolly, Anne Cleary, and Neil McKenzie developed devices that would enable their users to experience the world when seen through the eyes of animals. These "metaperceptual helmets" [Cleary et al., 2014] (Fig. 1.1) were inspired by the work of Stratton [1897] who, for an extended period of time, wore special glasses that would spatially invert his retinal images. During this peculiar self-experiment, he discovered that his visual system could eventually adapt to the new condition.

The ability of the human visual system to adopt a novel view is not only impressive in the light of this exercise. It is rather a crucial skill for successful social interaction. The term *perspective-taking* denotes an established concept in the field of psychology, describing the act of perceiving or understanding a situation from a different point of view. The aspect of *visual* perspective-taking is defined as the capability to understand how a scene is perceived by another person at a different position. Commonly, two levels of this skill are differentiated [Flavell et al., 1981]: Level-1 visual perspective-taking is the ability to estimate which objects in a scene are visible from a specific point of view, by essentially evaluating occlusion configurations

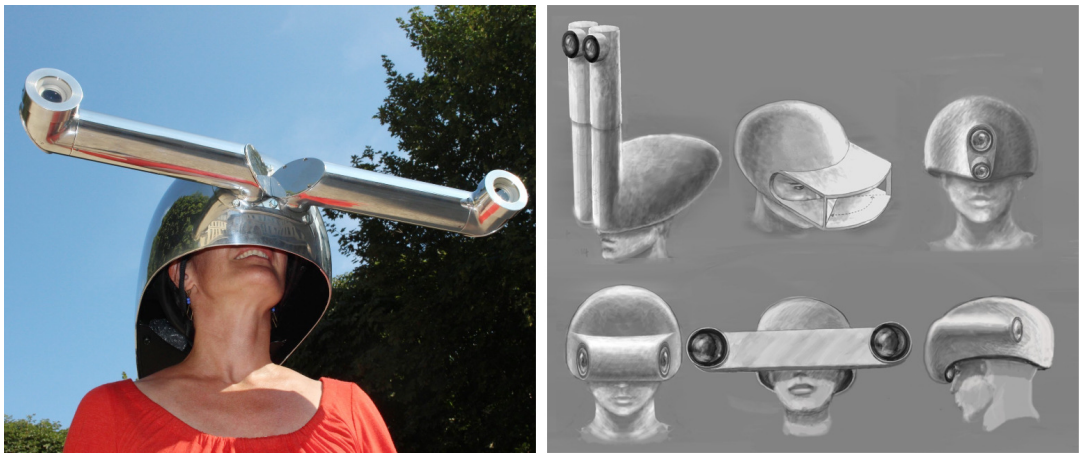


FIGURE 1.1: *"Metaperceptual helmets" allow the wearer to see the world from a different view.* Image courtesy of Cleary Conolly, used with permission.

in physical space. Level 2 deals with the more difficult question of how the objects in a scene are organized from the imagined point of view, e. g., to understand that an object to my right appears to be on the left for a person facing me. Comprehending what other people are able to see and how they perceive a situation is a core factor for successful social interaction [Gerace et al., 2013], ranging from a hide-and-seek game to warfare. It has also been shown to have strong links to creativity [Grant and Berry, 2011], which is not surprising when considering the amount of mental "in-painting" necessary to hallucinate unknown views.

Taking a novel view solely mentally is not easy. In fact, children acquire this skill rather late in their development. This is exemplified by the so called *three mountain problem* [Piaget and Inhelder, 1969]: Children were shown a model of three mountains of different appearance and a doll was placed in the scene. Now, the children were asked which photograph in a collection best reflected the doll's view. It turned out that only children above the age of seven could consistently choose the correct photograph. Studies in the field of comparative psychology indicate that higher animals exhibit signs of visual perspective-taking only scarcely [Itakura, 2004].

In the light of these and numerous similar findings, it becomes evident that taking a novel view requires a sophisticated amount of *intelligence*. This thesis argues that a machine rendering novel views can, analogously, highly benefit from *artificial intelligence* (AI).

In computer graphics, producing a novel view is seemingly straightforward: Just render a new image. However, using a classical rendering pipeline results in two fundamental problems. First, given finite computational capabilities, rendering an image can take a very long time. Evaluating physically-correct global light transport in a complex scene can require minutes to hours of computation. Finding a reasonable trade-off between quality and runtime performance is difficult. Second, a model of the scene to render might not exist. Sometimes, when we are interested in a novel view of a scene, all we have is an image.

Image-based rendering aims at solving these two problems. Here, a novel view is created just based on one or multiple already existing images (and possibly associated meta-information). Working with images instead of scene models reduces the computational load significantly and is very flexible at the same time. Image-based rendering naturally employs the redundancy between views: Moving a camera to the right basically shifts the image left. A straightforward image-based approach for novel-view synthesis is therefore to simply re-arrange pixels according to deformation rules. Consequently, image-based rendering usually excels when novel views are very similar to the original image(s).

However, efficiency is still a major concern in many applications. Frequently, interactive performance is a mandatory requirement. Virtual-reality applications demand a refresh rate of up to 90 Hz to prevent motion sickness. Other light field-related applications and distribution effect rendering require a multitude of views to be created or integrated. An established methodology for algorithm design in visual computing is to employ the massively parallel capabilities of modern graphics processing units (GPUs). Many rendering tasks are trivially parallel, e. g., operations are independent for pixels, and map well to the specific architecture of modern GPUs. However, working on mere images using massively parallel hardware is not a guarantee for satisfactory efficiency. In fact, the problem of interactive or real-time view synthesis is far from being solved and new ideas and algorithms are needed to tackle current and upcoming tasks.

This thesis provides novel techniques that increase the efficiency of image-based view synthesis algorithms. Recognizing and acknowledging that taking a novel

view is a complex task, it does so by injecting AI into several stages of the rendering pipeline, while at the same time retaining their property of being massively parallelizable. Unfortunately, the problems arising in image-based rendering are not offhandedly amenable to an intelligent agent. Rather, both a reformulation of the problems and custom intelligent algorithms are required to fully benefit from this joining. In successfully doing so, we are able to

- demonstrate major performance improvements for image-based view synthesis up to orders of magnitude, and
- extend the scope of manipulations possible with image-based methods by incorporating many design dimensions, like e. g., observer position, aperture size and shape, time and exposure interval, light source position, and wavelength of light.

This has two important consequences: Image-based rendering becomes (*i*) more efficient and (*ii*) more versatile. This has direct implications on the level of realism that is possible to achieve in interactive environments, such as computer games and visualizations, both on standard displays and in virtual or augmented reality. While research on traditional rendering techniques is slowly approaching a state of diminishing returns, the incorporation of more and more sophisticated AI technology appears to be a promising direction towards photo-realistic imagery in dynamic and interactive virtual environments.

1.2 Contributions

In this thesis we identify three core tasks of image-based rendering that benefit from certain kinds of intelligence: Point pattern design, distribution effect rendering, and 2D-to-3D conversion.

Point Pattern Design The design of point patterns is a fundamental task at the core of computer graphics and beyond, linking many topics such as light transport computation, layout, object placement, and visual perception. The use of correlated points as sampling patterns is a prevalent part of most rendering algorithms. In Chapter 3 (based on Leimkühler et al. [2019]) we contribute

- a GPU-friendly method to generate point patterns using a sequence of unstructured filters that work in high dimensions,
- a method that utilizes machine learning to determine these filters from prescribed design goals, and
- point patterns that have not been possible to create before.

Distribution Effects Complex light transport effects like depth of field, motion blur, soft shadows, and spectral dispersion are important factors to the cinematic quality of images. Adding these effects to images in a post-process is the topic of Chapter 4 and Chapter 5.

In Chapter 4 (published as Leimkühler et al. [2017]) we

- observe that distribution effects can be expressed as a sum of many pinhole images and contribute a warping-based synthesis framework to exploit the coherency among those images,

- introduce the notion of "distribution flow" that represents 2D image deformation in response to changes in relevant distribution coordinates, and
- use a planning algorithm to determine the traversal of the space of pinhole images to be synthesized, resulting in superior image warping performance.

We attack the problem of distribution effect rendering from another angle in Chapter 5 (published as Leimkühler et al. [2018a]), where we

- splat the point-spread functions (PSFs) of distribution effects into the image,
- accelerate this process by optimizing for a sparse representation of PSFs in the Laplacian domain, and
- demonstrate the feasibility and efficiency of this approach for a wide range of distribution effects.

2D-to-3D Conversion The conversion from monocular to binocular content is a prototypical image-based rendering problem and naturally requires a sophisticated amount of image analysis. In Chapter 6 (published as Kellnhofer et al. [2015], Leimkühler et al. [2016], and Leimkühler et al. [2018b]), we contribute

- a biologically-inspired real-time 2D-to-3D conversion system based on learned priors and depth cues,
- a probabilistic fusion procedure that takes different sources of evidence with varying degrees of confidence into account.

1.3 Outline

This thesis is structured as follows. Chapter 2 reviews relevant theoretical and practical background. In Chapter 3, we introduce a sampling algorithm that is based on machine learning. Chapter 4 demonstrates an image warping technique that gains its efficiency from a planning strategy. We move on to a distribution effect renderer in Chapter 5 that performs a sparsity-enforcing optimization in the space of point-spread functions. Next, a real-time 2D-to-3D conversion system is presented in Chapter 6 that combines ideas from machine learning, computer vision, and probabilistic reasoning. We conclude the thesis and discuss future research directions in Chapter 7.

Chapter 2

Background and Previous Work

In this chapter we provide a review of concepts, tools, and previous work relevant for this thesis. We recall basic mathematical concepts in Sec. 2.1. Necessary background in parallel computing for computer graphics tasks is given in Sec. 2.2, followed by a short review of human depth perception in Sec. 2.3. View synthesis constitutes the core topic of this thesis and its relevant preliminary aspects are discussed in Sec. 2.4. Sampling patterns are reviewed in Sec. 2.5, before Sec. 2.6 provides an overview of the field of artificial intelligence.

2.1 Mathematical Background

This section reviews fundamental mathematical concepts that are used throughout this thesis. All definitions are given for n-dimensional Euclidean space in Cartesian coordinates. Boldface fonts indicate vector-valued quantities.

2.1.1 Differential Operators

A differential operator is a mapping from a function f to another function which contains the derivative of f with respect to one or more variables. In the context of this thesis it is sufficient to only consider *linear* differential operators. The most basic operators are the derivative $\frac{\mathrm{d}}{\mathrm{d}x}$ itself and the partial derivative $\frac{\partial}{\partial x_i}$ for functions with multiple variables.

The gradient operator maps a scalar function of n variables to the vector of first partial derivatives, as in

$$\nabla : f \mapsto \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots \frac{\partial f}{\partial x_n} \right)^T .$$

The gradient vector points in the direction of the steepest ascent of f, while its magnitude corresponds to the slope of the graph of f in that direction.

The divergence of a vector-valued function is a scalar field expressing the vector field's volume density of outward flux, i. e., its source, and is given by

$$\mathrm{div} : \mathbf{f} \mapsto \sum_i^n \frac{\partial f_i}{\partial x_i}.$$

The divergence is positive in regions of local expansion, and negative in regions of local contraction.

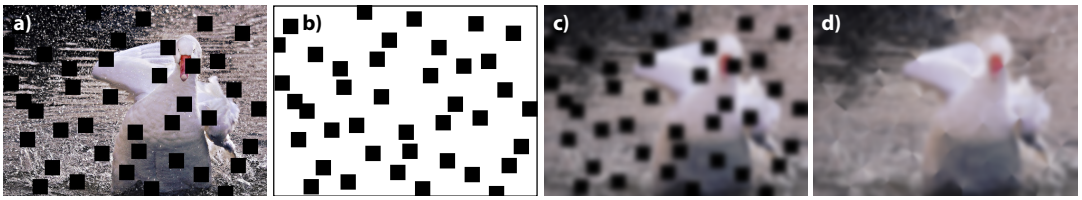


FIGURE 2.1: a) *Original image.* b) *Confidence image.* c) *Convolution of* a) *with a Gaussian function as defined in Eq. 2.1.* d) *Normalized convolution of* a) *with a Gaussian function as defined in Eq. 2.2 (after Knutsson and Westin, 1993).*

Finally, the Laplacian of a scalar function is given by

$$\Delta : f \mapsto \operatorname{div}(\nabla f) = \sum_{i}^{n} \frac{\partial^2 f}{\partial x_i^2}.$$

It is a straightforward generalization of the second derivative to functions with multiple variables. A useful property of the Laplacian is its rotational invariance, i. e., $R\Delta f = \Delta R f$, where R is an n-dimensional rotation.

2.1.2 Integral Transforms

An integral transform T is a mapping from a function f to another function of the general form

$$T : f(\mathbf{x}) \mapsto \int_{\Omega} f(\mathbf{u}) K(\mathbf{u}, \mathbf{x}) \mathrm{d}\mathbf{u}.$$

The integral kernel K uniquely characterizes the transform. The integration is carried out over an application-specific domain Ω. This thesis makes extensive use of integral transforms, in particular the ones described below.

Convolution

The convolution of two functions f and g is given by

$$(f * g)(\mathbf{x}) = \int_{\mathbb{R}^n} f(\mathbf{u}) g(\mathbf{x} - \mathbf{u}) \mathrm{d}\mathbf{u}. \tag{2.1}$$

$f * g$ can be interpreted as the weighted mean of f with the weights given by a mirrored version of g, or vice versa, due to the commutativity of the operation.

In some applications the function f features an associated confidence function c, indicating the amount of spatially-varying certainty of f. Integrating confidence into the convolution operator yields the normalized convolution operator [Knutsson and Westin, 1993] and is defined as follows:

$$(f *_c g)(\mathbf{x}) = \frac{(f \cdot c) * g}{c * g}(\mathbf{x}). \tag{2.2}$$

The denominator $Z := c * g$ is often referred to as partition function. Fig. 2.1 illustrates the concepts of convolution and normalized convolution.

Fourier Transform

The n-dimensional Fourier transform of a function f is defined as

$$\mathfrak{F}(f)(\mathbf{q}) = \int_{\mathbb{R}^n} f(\mathbf{x}) e^{-2\pi i \langle \mathbf{x}, \mathbf{q} \rangle} \mathrm{d}\mathbf{x}, \tag{2.3}$$

where $\mathbf{q}$ is an n-dimensional frequency, and $\langle \cdot, \cdot \rangle$ denotes the inner product of vectors. $\mathfrak{F}$ maps f into the complex-valued frequency domain, by performing a transform to the basis of sinusoids, as can be seen from Euler's formula

$$e^{ix} = \cos(x) + i \sin(x).$$

In many cases we are not interested in the full complex-valued result of the Fourier transform, but rather in the *power spectrum*, which corresponds to the magnitude of the frequency response and is simply given by the absolute value of $\mathfrak{F}(f)$.

In this thesis we exclusively deal with finite domains, and in particular the unit hypercube. This leads to the Fourier series representation

$$\mathfrak{F}(f)_{\mathbf{q}} = \int_{[0,1]^n} f(\mathbf{x}) e^{-2\pi i \langle \mathbf{x}, \mathbf{q} \rangle} \mathrm{d}\mathbf{x}, \tag{2.4}$$

where $\mathbf{q}$ is an n-dimensional integer Fourier series coefficient, i. e., frequency vector.

The non-uniform discrete Fourier transform of an unstructured n-dimensional point set $X = \{\mathbf{x}_k\}_{k=0}^{N-1}$ is given by

$$\mathfrak{F}(X)_{\mathbf{q}} = \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i \langle \mathbf{x}_k, \mathbf{q} \rangle}. \tag{2.5}$$

Please notice that Eq. 2.5 is a special case of Eq. 2.4, with f being a sum of shifted Dirac deltas.

Since the complex exponential function is periodic and discrete signals naturally have a finite extent, the discrete Fourier transform by design creates a periodic continuation of the signal both in the frequency and the primal domain.

Green's Function

Integral transforms can be used to invert differential operators (Sec. 2.1.1). Inverting the basic derivative operator is trivially done by integrating the derived function (i. e., $K = 1$). However, given the result y of a more complex differential operator L requires solving the inhomogeneous differential equation

$$Lf = y$$

for f. The integral transform that solves this equation depends both on the differential operator and the boundary conditions imposed. In this context, the integral kernel K is referred to as Green's function.

In this thesis we make use of the Laplacian operator for images. Therefore, we are naturally interested in the Green's function that solves the associated 2D Poisson problem, i. e., the transformation from the Laplacian to the primal domain. For the particular problem at hand we consider an infinite 2D domain with no boundary

conditions, which leads to the free-space Green's convolution kernel

$$K(\mathbf{u}, \mathbf{x}) = K(\|\mathbf{u} - \mathbf{x}\|_2) = \frac{1}{2\pi} \log(\|\mathbf{u} - \mathbf{x}\|_2 + \epsilon).$$

This kernel is radially symmetric and only depends on the distance $d = \|\mathbf{u} - \mathbf{x}\|_2$ to the origin. The value ϵ (we use $\epsilon = .5\,px$) prevents a singularity at $d = 0$.

Our images are compactly supported, that is, they are zero outside the unit square, which enforces enough constraints to make the integration solution unique. We use G to denote the operator applying the convolution with Green's function. This is routinely done in the Fourier domain [Bhat et al., 2008] or – even more efficiently – using a pyramidal scheme [Farbman et al., 2011]. A useful property is the rotational invariance of G, inherited from the rotational invariance of the Laplacian operator itself: integration of a rotated Laplacian yields the same results as the rotation of an integrated Laplacian.

In typical gradient image editing tasks, the manipulated [Bhat et al., 2010] or noisy [Lehtinen et al., 2013] gradient ∇f is given and a function f is to be found by employing the Laplacian as a means for finding a least-squares solution, often with additional regularization (screened). In this thesis, our methods never act on gradient vector fields ∇f, but directly on the scalar Laplacian Δf, allowing both sparse processing and accurate, yet efficient integration [Farbman et al., 2011].

2.1.3 Sparse Approximations

Functions can be represented in numerous different ways. Storing and processing functions is most efficient in a sparse representation, where only a few coefficients reveal the information we are interested in. The usual way of obtaining a sparse representation is to decompose the function of interest into elementary functions, called a *dictionary*. An orthogonal basis is an example of a dictionary of minimum size.

The core task of finding a sparse representation is to adapt the dictionary to the properties of the class of functions to be represented. In practice, however, it is oftentimes impossible to find a representation that fulfills certain sparsity constraints. Therefore, a common procedure is to obtain *sparse approximations*, which trade sparsity for accuracy. This can be done either by simply thresholding small dictionary coefficients [Donoho et al., 1992] or by directly optimizing for a desired sparsity-accuracy trade-off [Bach et al., 2011]. In this thesis, we make extensive use of a very specific kind of sparse approximation:

Scattered Samples

In a scattered data representation a function $\mathbf{f}(\mathbf{x})$ is stored as a list of sample positions and values $F = \{\mathbf{x}_k, \mathbf{f}(\mathbf{x}_k)\}_{k=0}^{N-1}$. This representation has the advantage of being able to adapt to local features of $\mathbf{f}$ by choosing appropriate densities. The design of sample patterns is a research question on its own and is discussed in Sec. 2.5. Obtaining $\mathbf{f}$ from F is referred to as *reconstruction*.

2.1.4 Radial Basis Functions

Radial basis functions (RBF) are a simple, yet powerful finite-element technique for reconstructing an unknown continuous function $\mathbf{f}$ from a sparse set of point samples $\{\mathbf{f}(\mathbf{x}_k)\}_{k=0}^{N-1}$ (Sec. 2.1.3). The core idea is to employ a real-valued function ϕ whose

TABLE 2.1: *Commonly used radial basis functions. The parameter a is an application-dependent scaling factor.*

Name	$\phi(r)$
Gaussian	e^{-ar^2}
Multiquadric	$\sqrt{r^2+a^2}$
Inverse Multiquadric	$\sqrt{r^2+a^2}^{-1}$
Thin Plate Spline	$r^2\ln(r)$

value solely depends on the distance to the origin, i. e., $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|_2)$. A weighted sum of RBFs shifted to the sample position $\mathbf{x}_k$ yields a continuous approximation of $\mathbf{f}$ of the form

$$\mathbf{f}(\mathbf{x}) \approx \tilde{\mathbf{f}}(\mathbf{x}) = \sum_{k=0}^{N-1} \mathbf{w}_k \phi(\|\mathbf{x} - \mathbf{x}_k\|_2),$$

where $\mathbf{w}_k \in \mathbb{R}^n$ are weights associated with the sample points. If the approximation is required to interpolate the sample points, i. e., $\mathbf{f}(\mathbf{x_k}) = \tilde{\mathbf{f}}(\mathbf{x_k})$, the weights can be determined by solving the linear system

$$\sum_{k=0}^{N-1} \mathbf{w}_k \phi(\|\mathbf{x}_j - \mathbf{x}_k\|_2) = \mathbf{f}(\mathbf{x}_j), \quad \text{for} \quad 0 \leq j < N.$$

To make sure that the linear system is solvable, RBFs are designed in such a way that the symmetric system matrix is positive definite. This holds for several standard RBFs, representative examples of which are given in Table 2.1.

2.1.5 Bayesian Statistics

Bayesian statistics provides the mathematical procedures to assess probabilities that are changing as new information is gathered. The central tool is Bayes' theorem: Given two events A and B, the conditional probability of A given B is computed as

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

While this expression is a fundamental result of probability theory and technically symmetric in terms of A and B, in the context of Bayesian statistics each term has a dedicated meaning. The event A usually represents a proposition whose *posterior* probability $P(A|B)$, given some evidence or new data B, we want to estimate. $P(A)$ is called the *prior* probability of the proposition, which expresses our knowledge about A without taking the evidence into account. $P(B|A)$ is called the *likelihood* and corresponds to the probability estimated from the evidence. Finally, $P(B)$ is the probability of the evidence, which serves as a normalizing factor (constant of proportionality).

A core task in the context of Bayesian statistics is the modeling of data-generating processes (Sec. 2.6.5). Maximum likelihood estimation (MLE) is the process of estimating the parameters of such a probabilistic model to maximize the likelihood function, i. e., to determine

$$B_{\text{MLE}} = \arg\max_B P(B|A).$$

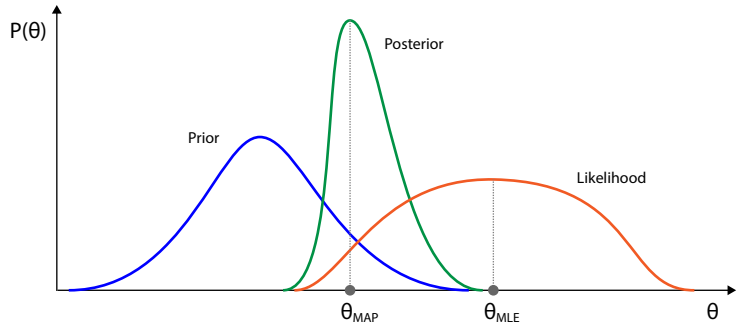


FIGURE 2.2: *Bayesian inference in a continuous one-dimensional domain.*

Analogously, the process of maximum a posteriori (MAP) estimation deals with finding parameters that maximize the posterior, i. e.,

$$A_{\text{MAP}} = \arg\max_A P(A|B).$$

Usually, $P(B)$ is not required for MAP estimation due to its independence of the model and therefore purely normalizing contribution. Figure 2.2 illustrates the concepts.

2.1.6 Monte Carlo Integration

Monte Carlo integration is a technique for numerically computing definite integrals of the general form

$$I = \int_\Omega \mathbf{f}(\mathbf{x}) \mathrm{d}\mathbf{x}.$$

Numerical integration is necessary whenever the integrand $\mathbf{f}(\mathbf{x})$ cannot be described analytically.

The basic idea of naive Monte Carlo integration is to uniformly sample $\mathbf{f}$ at positions $\{\mathbf{x}_k\}_{k=0}^{N-1}$. Due to the law of large numbers, for a sufficiently large N, the integral I can be approximated by

$$I \approx V \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{f}(\mathbf{x}_k),$$

where $V = \int_\Omega \mathrm{d}\mathbf{x}$ is the volume of Ω. When the sample positions $\mathbf{x}_k$ are chosen according to a probability density function $p(\mathbf{x})$, the Monte Carlo estimator generalizes to the form

$$I \approx \frac{1}{N} \sum_{k=0}^{N-1} \frac{\mathbf{f}(\mathbf{x}_k)}{p(\mathbf{x}_k)}.$$

The sampling density $p(\mathbf{x})$ can be chosen arbitrarily, as long as it is non-zero for all $\mathbf{x}$ for which $f(\mathbf{x})$ is non-zero. A common method to decrease the variance of the stochastic integral estimation is to design $p(\mathbf{x})$ in such a way that it matches the shape of $f(\mathbf{x})$ as closely as possible. This procedure is referred to as *importance sampling*.

The stochastic approach of Monte Carlo integration allows for an efficient computation in high-dimensional or even transdimensional domains, which are quite common in light transport computations (Sec. 2.4).

2.2 Parallel Visual Computing

All algorithms developed in this thesis gain a substantial amount of their efficiency from being massively parallel. This section therefore reviews the concepts of parallel computing for graphics applications.

2.2.1 Massively Parallel Computing

A GPU is a co-processor designed to perform massively parallel computations. Typical contemporary GPUs exhibit hundreds or thousands of cores [Barlas, 2014], allowing fine-grained parallelism. The prevalent compute model used is *single instruction, multiple data* (SIMD) [Flynn, 1972], i. e., a large collection of items is processed using the same instructions. This was motivated by graphics applications, where large sets of triangles and pixels are processed independently, resulting in trivially parallel tasks. Gradually, GPUs have been used in pipelines with a broader range of applications (general-purpose computing on GPUs, GPGPU), in particular in the field of scientific computing. Nowadays, many sophisticated interfaces (APIs) for programming GPUs are available, such as OpenGL [Shreiner et al., 2013], CUDA [Kirk, 2007], and OpenCL [Stone et al., 2010].

Optimization towards maximal GPU runtime performance requires considerations close to the hardware level [Steinberger, 2013] and involves e. g., avoiding thread divergence, efficient latency hiding, utilizing shared and other specialized memory, and partitioning computational blocks. This can include optimizations specifically tailored to dedicated pieces of hardware. Much of the above can be alleviated by an informed combination of problem formulation and data layout, that maps well to modern GPUs. Corresponding operations used throughout this thesis are discussed next.

2.2.2 Data-parallel Operations

The processors of modern GPUs are naturally arranged in and accessed via array structures. Therefore, data layouts in the form of implicit regular grids of relatively low dimensionality (typically ≤ 3) are beneficial. The resulting implicit connectivity between adjacent data points is pre-defined and does commonly not require sophisticated acceleration structures, which might facilitate thread divergence. The aggregation of data in regular grids is routinely performed using pyramidal structures, where fine-grained data is successively aggregated into coarser levels. This often results in constant amortized complexity per data point and has direct hardware support in the form of MIP mapping.

Most non-trivial parallel algorithms require a form of information exchange or message passing between data points. An example is convolution (Sec. 2.1.2), where neighboring points mutually contribute to their respective results. A parallel algorithm naturally has three options for realizing message passing: Parallelization over (*i*) senders, (*ii*) receivers, or (*iii*) messages. All strategies are equivalent and only differ in their execution efficiency.

In *gathering* algorithms, receivers iterate over potential senders. This corresponds to a search in the space of possible contributors to a result. When the data exhibits

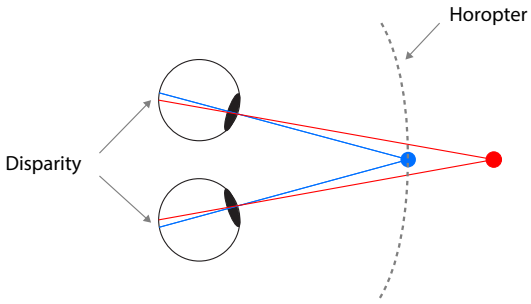


FIGURE 2.3: *The geometry of stereopsis. The eyes are converging to the blue point. This gives rise to binocular disparity for the red point, which lies outside the horopter surface.*

a regular layout and the search radius is relatively small, gathering is usually the more efficient procedure. In contrast, in a *scattering* algorithm, senders iterate over potential receivers. This procedure naturally requires incoherent memory writing and is therefore particularly efficient in settings where only sparse data needs to be transferred over possibly larger distances. The third option, the parallelization over interactions, is suitable for situations where a reasonably low number of messages is know a priori. This strategy can be beneficial when dealing with unstructured data, but it can result in inferior efficiency due to a combinatorial explosion in the number of messages. The decision for or against a particular parallelization strategy largely depends on the particular problem to be solved and consequently needs to be determined on a case-by-case basis.

A practical and efficient implementation of scattering is *splatting* [Westover, 1989]. Here, the rasterization pipeline is used to draw geometric primitives (e. g., points). The corresponding projected images trigger shading computations in the covered fragments.

2.3 Depth Perception

In this section we review the basic psychophysical and computational background of stereo vision and depth perception.

2.3.1 Stereopsis

Humans (along with many animals) obtain visual information from two eyes. Due to the lateral offset of the eyes, the two images projected to the corresponding retinas are slightly different. When a human observer looks at an object, the two eyeballs are rotated around a vertical axis so that the point of interest appears in the center of the retina in both eyes. In doing so, single vision is obtained. Closer objects result in an inward rotation (convergence) whereas distant objects lead to an outward rotation (divergence). The set of all points yielding single vision for a fixed vergence state is called *horopter*. Relative positional differences in the retinal images arising from the lateral offset of the eyes are commonly referred to as binocular *disparity* (Fig. 2.3). Since the amount of disparity depends on the distance to the horopter, the human visual system is able to infer depth by performing region matching.

2.3.2 Spatio-temporal Disparity Sensitivity

The spatial disparity sensitivity function determines the minimum disparity magnitude required to detect sinusoidal depth corrugations of various spatial frequencies [Howard and Rogers, 2012, Ch. 18]. The highest resolvable spatial frequency is about 3–4 cpd (cycles per degree), which is almost 20 times below the cut-off frequencies for luminance contrast [Wandell, 1995]. Similar investigations in the temporal domain indicate that the highest sinusoidal disparity modulation that can be resolved is about 6–8 Hz [Howard and Rogers, 2012], which is significantly lower than the 70 Hz measured for luminance [Wandell, 1995].

As analyzed by Kane et al. [2014], the picture is different for disparity step-edges in space and time, which are important in real-world images. They found that, for step-edge depth discontinuities, observers might still notice blur due to the removal of spatial frequencies up to 11 cpd, indicating that while overall disparity can be smoothed significantly, this is not the case for depth discontinuities. Kane et al. could further show that filtering temporal frequencies higher than 3.6 Hz from a step signal remains mostly unnoticed. Their findings indicate that the temporal disparity signal might be sparsely sampled and even more aggressively low-pass filtered, without causing visible depth differences.

Surprisingly, depth edges appear sharp, even though human ability to resolve them in space and time is low. One explanation for this is that the perceived depth edge location is determined mostly by the position of the corresponding luminance edge [Robinson and MacLeod, 2013]. Interestingly, depth discontinuities that are not accompanied by color edges of sufficient contrast poorly contribute to the depth perception and do not require precise reconstruction in stereo 3D rendering [Didyk et al., 2012].

The upper disparity gradient limit determines the maximum disparity for corrugations of a certain frequency the human visual system can fuse [Howard and Rogers, 2012, Fig. 18.28]. Intuitively, when increasing the disparity gradient (e. g., by slanting a surface), retinal images become dissimilar and fusion becomes impossible [Filippini and Banks, 2009]. Kane et al. [2014] generalize this observation to space-time.

2.3.3 Computational Models of Depth Perception

Inference of depth from monocular images is based on depth *cues*. A discussion of individual cues is beyond the scope of this thesis and can be found in [Howard and Rogers, 2012]. The combination of cues into a perception of depth is called *fusion*. If multiple cues are extracted, their computational fusion is considered difficult, and left to the user as in the system of Assa and Wolf [2007]. Two main opposing paradigms of fusion exist: the *weak* and the *strong* model [Landy et al., 1995]. In the weak model, cues act in isolation to produce an estimate of depth which is directly combined in a fixed linear weighting. In a strong model, cues interact in an unspecified and arbitrarily complex way.

A middleground is *modified weak* fusion [Landy et al., 1995], in which cues are independent, but their combination is not a linear mixture with fixed weights, as it locally adapts to the confidence of each cue. Bayesian fusion [Knill and Richards, 1996] using normal distributions is a formal way to achieve modified weak fusion. Here, cues are weighted by their confidence before they are combined. Besides using only the cues of the present stimulus, one strength of Bayesian inference is that it can account for prior experience (Sec. 2.1.5).

FIGURE 2.4: *Different types of distribution effects.* (a) *Depth of field.* (b) *Motion blur.* (c) *Spectral caustics.* (d) *Soft shadows.*

2.4 View Synthesis

This section reviews the theoretical and practical background of view synthesis with an emphasis on the topics relevant for this thesis.

2.4.1 Light Transport and Rendering

Synthesizing views means simulating light transport. The rendering equation [Kajiya, 1986] concisely describes how radiance L of wavelength λ leaving a differential surface patch at location $\mathbf{x}$ with normal vector $\mathbf{n}$ in the direction $\boldsymbol{\omega}_o$ at time t can be computed:

$$L(\mathbf{x}, \boldsymbol{\omega}_o, \lambda, t) = L_e(\mathbf{x}, \boldsymbol{\omega}_o, \lambda, t) + \int_{S^+} L(\mathbf{x}, \boldsymbol{\omega}_i, \lambda, t) f_r(\mathbf{x}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_o, \lambda, t) \langle \boldsymbol{\omega}_i, \mathbf{n} \rangle \, \mathrm{d}\boldsymbol{\omega}_i. \tag{2.6}$$

Here, L_e is emitted radiance, S^+ the upper hemisphere above $\mathbf{x}$, $\boldsymbol{\omega}_i$ an incoming direction, and f_r is the bidirectional reflectance distribution function (BRDF) encoding surface properties. As light arriving from a direction $\boldsymbol{\omega}_o$ recursively depends on all incoming light reflected into this direction, this integral equation has highly non-trivial solutions even for the simplest geometric configurations.

The function that describes radiance "flowing" in every direction at every point in space is called *light field* [Gershun, 1939]. Neglecting spectral and temporal effects, this plenoptic function [Adelson and Bergen, 1991] is naturally five-dimensional (3 positional and 2 angular dimensions), but since radiance is constant in free space the effective parameter space reduces to a four-dimensional manifold [Gortler et al., 1996].

Distribution Effects

Many phenomena arising from physical light transport cannot be computed by a mere point-wise evaluation of Eq. 2.6, i. e., a single parameter vector $(\mathbf{x}, \boldsymbol{\omega}_o, \lambda, t)^T$. Rather, an integration of light contribution over one or multiple variables is necessary. The degree of realism of synthetic images heavily depends on the successful simulation of these *distribution effects*.

Depth of Field Images captured with physical cameras have a limited depth of field (DoF) (Fig. 2.4, a). Objects further away from the focal plane exhibit stronger defocus blur. This is, as light arrives at each sensor location from multiple directions, namely from the entire lens surface, which corresponds to an integration over the domain of incoming directions, i. e., $\boldsymbol{\omega}_o$.

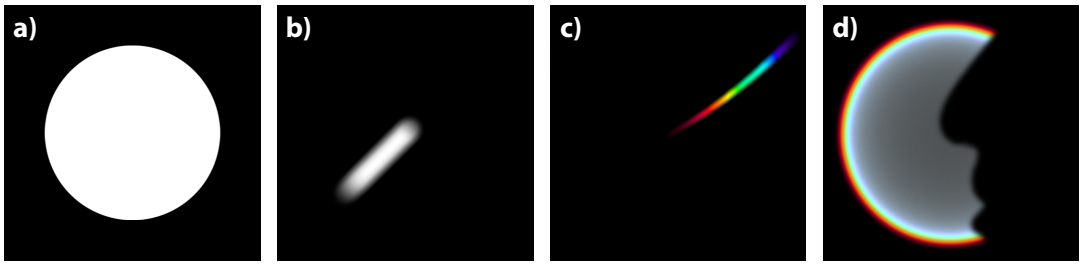


FIGURE 2.5: *Different types of point-spread functions.* (a) *A circular shape arising from the depth-of-field of a thin-lens camera.* (b) *A response exhibiting linear motion-blur and depth of field.* (c) *A spectral response arising from a chromatic aberration.* (d) *A partially occluded depth-of-field response of a physical lens model.*

Motion Blur Real cameras do not capture incoming light instantaneously. Photons are rather accumulated within a finite shutter interval, which corresponds to an integration over the temporal domain, i. e., t. In the case of moving objects or a moving camera, this leads to motion blur (MB) along the motion trajectory (Fig. 2.4, b).

Spectral Effects When light hits a dispersive interface like e. g., glass, it gets refracted. The new ray directions depend on the wavelength of the light, which oftentimes creates colorful effects. This phenomenon corresponds to an integration over the spectral domain, i. e., λ, and is observable either as *caustics*, where dispersed light hits a diffuse surface (Fig. 2.4, c), or when a scene is directly observed through a dispersive medium, e. g., a lens producing chromatic aberrations.

Soft Shadows In contrast to the facilities of virtual scenes, real light sources always have a physical extent. This naturally creates soft shadows, whose intricacies depend on the geometric configuration of light sources, occluders and shadow receivers (Fig. 2.4, d). These effects are determined by integrating over the surface of the light source. While Eq. 2.6 handles this situation naturally, soft shadows are nevertheless listed here, as their practical computation strongly resembles the ones of the other distribution effects mentioned above.

Most distribution effects can be understood in terms of their *point-spread functions* (PSF) [Kolb et al., 1995]. A PSF describes the response of the distribution effect integration to a single point. This response can become arbitrarily complex, as it naturally arises from e. g., perspective projection, occlusion, or dispersion (Fig. 2.5). A formal definition of PSFs is given in Sec. 5.3.1.

Computer graphics has a long history in modeling the imperfections of physical lens systems and film exposure to the end of providing the desired cinematic fidelity of real imagery. The canonical way of simulating distribution effects is distribution ray-tracing [Cook et al., 1984; Pharr et al., 2016] in combination with proper camera models [Kolb et al., 1995], which employs Monte Carlo integration: For every sample on the lens, in time, etc., a ray is sent. The final image is obtained by averaging the contribution of all rays. This is a general and easily implemented solution, but inherits the difficulties of ray-tracing that does not deliver the same performance as rasterization. Despite its theoretical properties, ray-tracing does not scale favorably with increasing geometry. Also shading complexity directly affects the cost of distribution effects as it is usually not able to exploit the coherence in shading, which is very similar in many cases. However, efficient reuse of samples is possible

by considering anisotropic footprints [Lehtinen et al., 2011], allowing to reduce the sampling rate significantly.

An alternative to ray-tracing for high-quality images is REYES [Cook et al., 1987]. Here, the scene is decomposed into micro-polygons, shaded and rasterized from many views. The shading of vertices decouples it from visibility determination, allowing to exploit coherence. While GPU implementations of REYES are available [Zhou et al., 2009], the complexity of the process does not map as well to GPUs as plain rasterization does.

Accumulation buffering [Haeberli and Akeley, 1990; Yu et al., 2010] is a simple alternative to this: A GPU is used to render many slightly different pinhole images, which is a highly optimized process taking full advantage of common graphics hardware. The resulting images are accumulated to produce an image with distribution effects. While this is a fairly simple and general method, like every super-sampling it can be slow and the (shading) coherence between similar images is not exploited.

Instead of averaging many pinhole images, an alternative solution is to blend the random nature of distribution ray-tracing and rasterization in what is called stochastic rasterization [Akenine-Möller et al., 2007]. Here, an output image is still rasterized, but contains pixels from many views which are then aggregated to support many distribution effects. Using layered rendering improves the quality here [Munkberg et al., 2014]. Other specialized rasterizations have directly produced multiple views [Hasselgren and Akenine-Möller, 2006]. Finally, direct samples of a light field rendering, i. e., multiple views, can be produced and aggregated using optimization for natural image (or light field) sparsity [Heide et al., 2013]. The optimization and the required ray-tracing have not been demonstrated with interactive performance and are limited to the lens domain.

While the algorithms mentioned above are to some extend general solutions for rendering distribution effects and beyond, specialized solutions, such as Lee et al. [2009] for depth-of-field do perform better. However, it is not clear how those approaches should be combined, and what their resulting performance would be. Also they often make assumptions on motion, scene configurations, sensor shapes etc. that might not apply in general.

The reconstruction of noise-free images from stochastic images has received substantial attention [McCool, 1999; Kontkanen et al., 2006; Lehtinen et al., 2011; Sen and Darabi, 2012]. In particular for DoF and MB, light transport Fourier analysis [Durand et al., 2005; Egan et al., 2009; Soler et al., 2009; Belcour et al., 2013; Munkberg et al., 2014; Yan et al., 2015b] is of great importance. Such approaches account for the effect of DoF and MB as a filter that reduces the bandwidth and therefore allows blurring the image, eventually also reducing MC noise. However, the derivations often consider only a limited set of geometric relations, distance assumptions, or diffuse surfaces.

2.4.2 Image-based Rendering

The concepts and algorithms described in the last section are concerned with synthesizing images from physical first principles or approximations thereof. This requires the creation of virtual scenes, including geometric modeling and the definition of surface properties and lighting configurations. A different procedure for synthesizing views is the manipulation of existing images. This *image-based rendering* approach is not concerned with solving the rendering equation (Eq. 2.6) explicitly, but rather aims at re-using pixels from images similar to the one we want to create, possibly including additional information like depth or surface orientation.

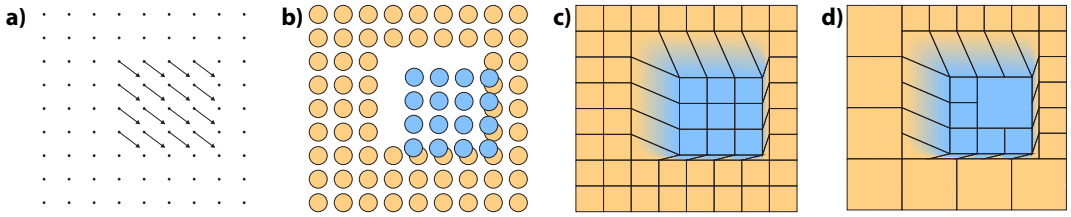


FIGURE 2.6: *Image warping.* (a) *Deformation flow field, resulting in occlusions and disocclusions.* (b) *Point warping.* (c) *Grid warping.* (d) *Quad-tree warping.*

Image-based rendering has two tremendous advantages compared to classical model-based rendering: First, the computations involved in manipulating pixels, i. e., regular two-dimensional arrays, are substantially cheaper. The reduced amount of information allows for efficient algorithms and the regular structure of the data maps well to GPUs. Second, image-based rendering is not reliant on scene descriptions. In fact, regular images taken by real cameras can be used as a basis for novel-view synthesis, substantially broadening the scope of this approach.

The main disadvantage of image-based rendering is the natural scarcity of information. Only scene content visible in the source image is available. Therefore, occlusions are a typical problem that needs to be tackled. Furthermore, the uniform sampling in image space implies a highly non-uniform sampling in world space: Sampling density is low for distant objects and slanted surfaces, which can be problematic in a local *magnification* configuration, where novel views require objects coming closer or surfaces rotating into view. Finally, view-dependent shading information is lost. Non-Lambertian surfaces change their appearance depending on the viewing angle. This effect is considered hard to accommodate.

Due to the discussed limitations, image-based rendering is typically used in applications where views similar to the original image are to be produced. In this restricted application domain, the advantages tend to outweigh the disadvantages.

Warping

The deformation of an image is commonly referred to as *warping* [Mark et al., 1997]. Input to a warping algorithm are an image and a *flow field*, indicating where each pixel should be moved (Fig. 2.6, a). Commonly, flow fields are non-injective and non-surjective, i. e., many pixels can move to the same position and some positions are never mapped to. The first situation corresponds to an occlusion and per-pixel depth information is needed in addition to the two-dimensional flow vector to resolve it. The second situation corresponds to a disocclusion, which either needs additional information from another image [Shade et al., 1998] or an in-painting strategy. Crucially, no obvious way exists to determine these kinds of visibility configurations induced by a general flow field without executing the warping. A flow field can originate from a simple reprojection or problem-specific deformation rules (Sec. 4.3.2).

Deforming images to produce novel views has its origin in interactive exploration frameworks [Lippman, 1980], where images undergo simple transformations for more appealing transitions. These basic ideas were further developed in the vision community, where extrapolation from given real images is often required as no mechanism exists to "render" arbitrary novel views [Chen and Williams, 1993]. Later, the opposite was exploited: what is possible for real images is possible for synthetic ones, too. Mark et al. [1997] were the first to present efficient image deformation to speed up rendering.

When used in interactive applications such as games, computational efficiency of warping has become important. Early graphics systems [Torborg and Kajiya, 1996] have warped entire tiles instead of re-rendering using a linear mapping. If tiles cover non-planar geometry, artifacts arise. Plain drawing of points (Fig. 2.6, b) is neither fast nor does it give high quality: Holes naturally appear in the presence of disocclusions or magnification. Drawing connected quads (Fig. 2.6, c) is a remedy, but, due to the high number of primitives, does not deliver the performance required for many warps. To reduce the number of primitives, quad trees were used [Didyk et al., 2010a] (Fig. 2.6, d). In all cases, using layered depth images [Shade et al., 1998] can improve quality [Widmer et al., 2015].

The best performance can be achieved when warping is expressed as a gathering instead of a scattering operation [McMillan, 1997; Yang et al., 2011; Bowles et al., 2012]. Here, instead of mapping pixels to a new location along a forward map, the inverse of the map is found and used to decide from which position a pixel in the output image is read. In practice, such approaches can require many iterations and diverge if not initialized properly, such as by using a forward warp.

Image-based Distribution Effects

An efficient way to simulate distribution effects is to start with a pinhole image and to apply suitable post-processing operations. This means that the involved integral computations are approximated by an informed utilization of pixel data alone.

Simple image blurring [Potmesil and Chakravarty, 1981] has been popular for DoF and MB and found commercial use in practical applications such as games [Göransson and Karlsson, 2007]. Using image-space filters has been proposed in many different forms [Rosado, 2007; McGuire et al., 2012]. A typical DoF solution is to use MIP fetches [Kraus and Strengert, 2007] or learned kernels in a neural network [Nalbach et al., 2017]. While very fast, these methods use convolution and are only correct for translation-invariant PSFs.

An alternative approach is to splat PSFs [Lee et al., 2008]. Here, for each pixel in the image the corresponding PSF is determined and additively drawn. Ray-marching (layered) depth images [Lee et al., 2009; Lee et al., 2010] is another method to produce high-quality DoF at high speed. They support complex DoF, producing a cinematic effect. However, they still result in MC noise and we are not aware of extensions to MB.

Warping-based distribution effects have been explicitly demonstrated for depth-of-field [Yu et al., 2010], but most warping-based work considers distribution effects as a common test case for the performance they achieve [Mark et al., 1997; Yang et al., 2011; Bowles et al., 2012; Didyk et al., 2010a]. Related applications of warping are light field pre-filtering [Zwicker et al., 2006], multi-view expansion [Didyk et al., 2013], temporal up-sampling, specular rendering [Lochmann et al., 2014] or even spectral rendering [Elek et al., 2014].

2D-to-3D Conversion

The conversion from monocular to binocular images is a core discipline in the field of image-based rendering. This task naturally is a two-step procedure: First, binocular disparity needs to be estimated from the monocular image. The second step produces the binocular image pair. Here, we review manual and automatic approaches for 2D-to-3D conversion with an emphasis on real-time conversion, along with the use of luminance and depth edges in computational stereo.

Disparity Estimation Manual conversion produces high-quality results but requires human intervention, which can result in substantial cost. It is based on painting depth annotations [Guttmann et al., 2009] with special user interfaces [Ward et al., 2011] and propagation in space and time [Lang et al., 2012]. The semi-supervised method of [Assa and Wolf, 2007] combines cues extracted from an image with user intervention to create depth parallax.

Automatic conversion does not induce manual effort, but results in long computation times to produce results of medium quality. The system of Hoiem et al. [2005] infers depth from monocular images by a low number of labels. Make3D [Saxena et al., 2009] is based on learning appearance features to infer depth. This approach shows good results for static street-level scenes with super-pixel resolution but requires substantial computation. Non-parametric approaches rely on a large collection of 3D images [Konrad et al., 2012] or 3D videos [Karsch et al., 2014] that have to contain an exemplar similar to a 2D input. Conceptually, such an approach aligns all 3D images or 3D videos in a large collection (hundreds of exemplars) with a monocular query input image or video and transfers their depth to the query. Aligning to a large collection of images or videos of hundreds of elements usually contradicts real-time requirements. For cel animations, where each frame is drawn manually and therefore usually contains pronounced outlines, T-junctions have been shown to provide sufficient information to add approximate depth [Liu et al., 2013]. Tao et al. [2013] estimate depth by fusing information obtained from defocus and correspondences in a confidence-aware fashion using a Markov Random Field. Their offline method requires full light fields.

Real-time methods to produce disparity from 2D input videos usually come at low visual quality. Individual cues such as color [Cheng et al., 2010], motion [Huang et al., 2009] or templates [Yamada and Suzuki, 2009] are combined in an ad-hoc fashion. A simple and computationally cheap solution is to time-shift the image sequence independently for each eye, such that a space-shift provides a stereo image pair [Murata et al., 1998]. This requires to identify the camera velocity and only works for horizontal motions. For rigid motions in animations, structure-from-motion (SfM) can directly be used to produce depth maps [Zhang et al., 2007]. Classical SfM makes strong assumptions about the scene content such as a rigid scene with camera motion. More recent work relaxes these assumptions [Vogel et al., 2015], but comes along with high computational costs. Commercial 2D-to-3D solutions [Zhang et al., 2011] based on custom hardware (e. g., JVC's IF-2D3D1 Stereoscopic Image Processor) and software (e. g., DDD's Tri-Def-Player), reveal little about their used techniques, but anecdotal testing shows the room for improvement [Karsch et al., 2014].

Recently, depth estimation based on deep learning outperformed most previous approaches. This class of algorithms can be trained either in a supervised [Eigen et al., 2014] or an unsupervised [Garg et al., 2016] fashion. Oftentimes it proved beneficial to incorporate a probabilistic model into the system [Li et al., 2015; Liu et al., 2016; Roy and Todorovic, 2016] to obtain high-resolution results and enforce an agreement between depth and luminance edges, as discussed next.

Depth and Luminance Edges Since luminance and depth edges often coincide, e. g., at object silhouettes, full-resolution RGB images have been used to guide depth map upsampling both in the spatial [Kopf et al., 2007] and the spatio-temporal [Richardt et al., 2012] domain. An analysis of a database with range images for natural scenes reveals that depth maps mostly consist of piecewise smooth patches separated by edges at object boundaries [Yang and Purves, 2003]. This property is used in depth compression, where depth edge positions are explicitly encoded, e. g.,

by using piecewise-constant or linearly-varying depth representations between edges [Merkle et al., 2009]. This in turn leads to a significantly better depth-image-based rendering quality than is possible at the same bandwidth of MPEG-style compressed depth, which tends to blur depth edges.

Differential and incremental image changes

This thesis make use of differentials and incremental changes in images. This section therefore briefly reviews related concepts.

Capture Epsilon photography [Raskar, 2009] refers to techniques in computational photography that rely on multiple images acquired by incrementally varying camera parameters like aperture, exposure, or viewpoint. Observing that there is much redundancy in the acquired images, compressive epsilon photography [Ito et al., 2014] only acquires a subset of the images and uses techniques from compressive sensing to infer the missing information.

Synthesis Ray and path differentials [Igehy, 1999; Suykens and Willems, 2001] describe the spatial change of a ray under a differential change of the associated sensor coordinate. This quantity essentially corresponds to the ray's footprint and is therefore an effective means for anti-aliasing. This concept was extended to also include the spectral domain [Elek et al., 2014]. The differentials are effectively analytic models for computing the change of ray properties with respect to a limited set of parameters.

The gradient domain has previously been used for image processing [Heckbert, 1986; Simard et al., 1999; Pérez et al., 2003; Bhat et al., 2010], and recently received new interest in contexts such as image compression [Galić et al., 2008], vector graphics [Orzan et al., 2013], reprojection [Kopf et al., 2013], texture representation [Sun et al., 2012], and realistic image synthesis [Lehtinen et al., 2013]. In gradient-domain path tracing [Kettunen et al., 2015] pairs of paths are traced for explicitly estimating image gradients. In conjunction with a Poisson reconstruction step this method yields superior results compared to standard path tracing.

2.5 Sampling

While random sampling is undoubtedly useful for many computer graphics tasks, it remains an active topic of research to determine exactly which form of randomness is desirable. In this section, we review different properties of point patterns along with algorithms to create them. An in-depth survey is provided by Yan et al. [2015a].

2.5.1 Pattern Properties

Yellott [1983] first noted that the receptors on the retina are neither regular nor random but follow very specific patterns where they keep a minimal distance. These patterns are routinely characterized by their expected *power spectrum* [Ulichney, 1988; Lagae and Dutre, 2008]. The expected power spectrum of a point pattern is computed by averaging over the Fourier transforms (Sec. 2.1.2) of many instances (Fig. 2.7, a and b). For two or more dimensions, the full spectrum is often further radially averaged to a one-dimensional subspace (Fig. 2.7, c). The variance of this radial estimate is called

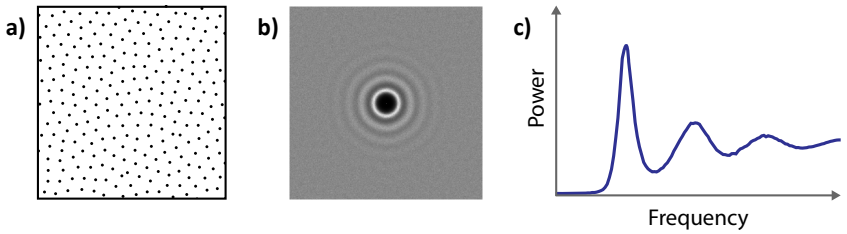


FIGURE 2.7: *A sampling pattern* (a) *is characterized by its expected power spectrum* (b) *that can be radially averaged to obtain a one-dimensional profile* (c). *This example shows a blue noise sample pattern, as can be seen from the vanishing energy in the low-frequency region.*

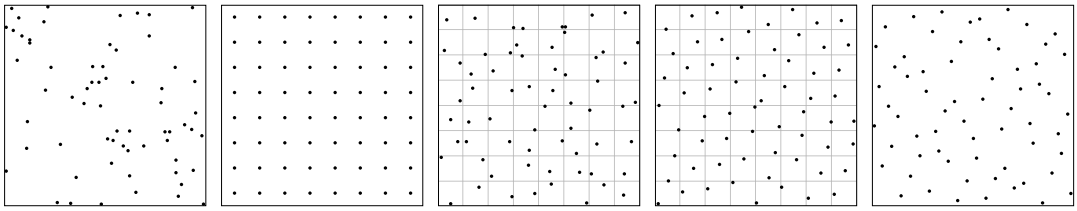


FIGURE 2.8: *Different classical point patterns in 2D.*

the radial *anisotropy* which is low for radially symmetric (isotropic) patterns and large for others.

Spectral properties of point patterns are often described in terms of noise "colors". A *white noise* pattern has a flat spectrum, where the energy is equally distributed over all frequencies. A *blue noise* (BN) pattern has a power spectrum with little energy in the low-frequency region. BN was first used in graphics for dithering [Ulichney, 1988] and stippling [Secord, 2002; Oliver et al., 2001]. In the context of dithering, models of human perception can be used to improve quality [Mulligan and Ahumada, 1992]. BN patterns are also used for Monte Carlo integration-based image synthesis (Sec. 2.1.6), as they shift the error into the high-frequency bands, to which humans are less sensitive [Cook, 1986]. Besides BN, other colors of noise are useful in tasks such as procedural primitive placement.

Heck et al. [2013] were the first to address oscillations in the power spectrum. Kailkhura et al. [2016] suggest to add a stair to the spectrum to widen the low-energy region in the low frequencies.

Another concept to create BN is the *Poisson disk* [McCool and Fiume, 1992; Lagae and Dutre, 2008] or max-min distance. In such a pattern, the minimal distance from one point to the others is maximized over all points i. e., all points keep a minimal distance.

Histograms of points distances (the *differential domain*) [Bowers et al., 2010; Wei and Wang, 2011] or pair correlations [Öztireli and Gross, 2012] are an alternative and flexible tool to analyze point patterns. In particular, they allow working on non-uniform point patterns and anisotropic spectra [Singh and Jarosz, 2017].

2.5.2 Pattern Generation

The most simple way to produce an n-dimensional point pattern is *uniform random* sampling (Fig. 2.8, a). However, the white noise power spectrum of this pattern naturally leads to clusters and holes.

The exact opposite of uniform random sampling is *regular* sampling (Fig. 2.8, b), which deterministically places points on a uniform grid. This method has two main

drawbacks: First, it produces regularity artifacts that are clearly observable in many applications. Second, it suffers from the *curse of dimensionality*: The number of points required to fill a domain grows exponentially with the number of dimensions.

A straightforward method to attack the first problem is to mix the two patterns described above. This leads to *jittered* sampling [Cook, 1986] (Fig. 2.8, c), where a uniform random offset is applied to a regular sampling pattern. This results in a stratified result, where only at most 2^n samples can clump together. The quality of jittered sampling can be further improved by stratifying the random offset applied to each sample [Chiu et al., 1994] and shuffling those offsets in a correlated way [Kensler, 2013]. The result is referred to as *correlated multi-jittered* sampling (Fig. 2.8, d).

Quasi Monte-Carlo (QMC) sampling denotes a family of sampling techniques that is based on low-discrepancy sequences. The governing principle of these deterministic methods is the representation of numbers in prime bases and their radical inverses. Popular low-discrepancy sequences are the Halton [Halton, 1964] (Fig. 2.8, e) and the Sobol [Sobol, 1994] sequence.

Classic ways to produce BN patterns are dart throwing [McCool and Fiume, 1992] and Lloyd relaxation [Lloyd, 1982]. The first can be slow, while the latter often suffers from regularity artifacts, that need extra effort to be overcome [Balzer et al., 2009; De Goes et al., 2012].

Methods that generate point patterns with a desired spectrum need to be run every time when a point pattern is produced [Wei and Wang, 2011; Zhou et al., 2012; Heck et al., 2013; Kailkhura et al., 2016]. Mitchell et al. [2018] have recently suggested methods to produce BN point sets in high dimensions, including point saturation.

Many technical alternatives have been considered to produce blue noise patterns such as variational [Chen et al., 2012], optimal transport [Qin et al., 2017; De Goes et al., 2012], tiling [Ostromoukhov et al., 2004; Wachtel et al., 2014], Wang tiles [Kopf et al., 2006], kernel-density estimation [Fattal, 2011], smooth particle hydro-dynamics [Jiang et al., 2015] or electro-statics [Schmaltz et al., 2010]. All these methods include involved mathematical derivations, can only realize a subset of properties and are limited in dimensionality and/or speed.

2.5.3 Projective Subspaces

Several methods try to explicitly produce patterns with desired spectra also in their projections. Chiu et al. [1994] proposes a construction that is jittered in 1D as well. Reinert et al. [2016] produce n-D BN patterns that retain BN also when projected to subspaces. This results in a typical cross-like spectrum. Ahmed et al. [2016] suggest a way to produce 2D patterns that are BN, but also low-discrepancy. Perrier et al. [2018] contribute an efficient and progressive point sequence to combine BN and LD using tiling. Singh and Jarosz [2017] show convergence improvements by shearing the subspaces according to the integrands' spectra. Production renderers [Kulla et al., 2018] also notice improvements when mixed subspaces are properly handled.

The relation of blue noise, low-discrepancy and variance reduction in MC integration is not fully clear [Christensen et al., 2018; Dobkin et al., 1996; Mitchell, 1992; Shirley, 1991]. It is evident, however, that there are methods that result in a low error, yet produce a more suspicious artifact pattern and that there are other approaches that produce visually pleasing patterns at the cost of a high error [Georgiev and Fajardo, 2016].

2.6 Artificial Intelligence

Artificial intelligence is a big field [Russell and Norvig, 2016] and continues to grow at a fast pace. This section first provides an overview of the field, before diving into the specific concepts and tools relevant for this thesis.

2.6.1 Overview

The field of artificial intelligence (AI) is concerned with the automation of intelligent behavior. While "intelligence" is a notoriously ill-defined term, an operable definition of what AI *does* can be formulated as follows: The analysis of a task followed by actions to successfully and efficiently fulfill it. Autonomous devices that exhibit artificial intelligence are referred to as *agents*. Agents act rationally, i. e., they are driven by achieving the best or best expected outcome. This outcome is formalized as a measure of performance, which can be defined explicitly as a utility function (e. g., an energy or loss to minimize) or implicitly (e. g., by observing that an item has not yet been found).

Agents receive input from and act in an *environment*. Types of environments differ vastly, ranging from the real world, over a game of chess, to the space of light field samples. This extremely broad range of applications and the resulting impact qualifies AI as a general purpose technology [Bresnahan and Trajtenberg, 1995], effectively categorizing it among e. g., electricity and medicine. While AI is naturally associated to the field of computer science, it draws upon a diverse set of disciplines, including mathematics, control theory, neuroscience, psychology, linguistics, philosophy, and economics.

Historically, the field was founded to develop the necessary technology to mimic human cognitive abilities. This so called *strong AI* hypothesis, sometimes even including goals as difficult as consciousness, was successively weakened, up to the point where AI research split up into numerous subfields, each tackling isolated narrow problems (*weak AI*). With the advent of recent technologies, however, the borders between these subfields tend to gradually soften and the idea of general intelligence as a long-term goal regains interest [Baum, 2017]. Critical voices argue that strong AI is technically impossible [Penrose, 1999], or bears undesirable social and ethical implications [Russell et al., 2015], up to an existential thread to humankind.

As AI applications become more and more mainstream, tasks previously considered to require intelligence tend to be excluded from the definition of AI in public discourse. This so called "AI effect" effectively reduces the field to the set of yet unsolved problems [McCorduck, 2009]. In this thesis we do not adapt this view and consider AI techniques and tools intelligent, even if they are historical achievements of the field.

In the following paragraphs we provide short descriptions of the central subfields of AI. More in-depth reviews of the particular tools and concepts relevant for this thesis are given in the sections thereafter.

Reasoning Generating conclusions from knowledge is at the very core of AI. Classical reasoning systems use variations of propositional or predicate logic to perform inductive or deductive inference. Probabilistic reasoning (Sec. 2.6.5) performs inference under uncertainty, using tools like Bayesian statistics (Sec. 2.1.5) or fuzzy logic.

Planning This branch is concerned with the development of strategies to drive sequences of actions. Classical planning (Sec. 2.6.3) operates in fully observable, static environments, which are represented by discrete state spaces. More advanced planning problems include factors such as partially observable and dynamic environments with stochastic elements and possibly multiple agents (inter-)acting in it.

Learning A learning agent improves its performance by observing its environment (Sec. 2.6.4). A mathematical model with modifiable parameters is optimized by considering environment samples, known as training data. The benefit of machine learning is that the system does not need explicit instructions in the form of "hand-crafted" algorithms, but rather utilizes general-purpose learning algorithms to automatically find patterns in the training data most useful for the task it tries to accomplish.

Perceiving Machine perception is the automated interpretation of raw sensor information. Sensor modalities can range from simple inputs like a switch to sophisticated pieces of hardware like multi-spectral cameras. Computer vision is one of the most prominent sub-disciplines and is concerned with the acquisition, processing, and analysis of digital images. The range of tasks includes low-level problems such as edge detection and motion estimation, as well as much more high-level challenges such as scene reconstruction or semantic action classification.

Natural Language Processing Since humans, different from computers, communicate using natural languages, an important contributor to successful human-computer interaction is the development of computer systems that understand natural languages. Common tasks include the analysis of syntax and semantics, the recognition and synthesis of speech, as well as information extraction from human-written sources.

Representing Knowledge As soon as a non-trivial amount of knowledge needs to be inserted into an intelligent system, a proper representation of information becomes crucial. Expert systems deal with tractable collections of knowledge required for narrow domains, whereas a comprehensive collection of commonsense knowledge requires a structured representation of concepts like objects, properties, time, causal relationships, and knowledge itself.

2.6.2 Optimization

Mathematical optimization is a general-purpose tool useful for many disciplines. Many AI algorithms employ concepts from this branch of applied mathematics. Continuous optimization refers to the theory and techniques of finding the minimum value $\hat{\mathbf{x}}$ of an objective function $f(\mathbf{x})$, i. e.,

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} f(\mathbf{x}),$$

with $f \in \mathbb{R}^n \to \mathbb{R}$ and $\hat{\mathbf{x}} \in \Omega \subseteq \mathbb{R}^n$, where Ω is the possibly constrained domain of feasible solutions. *Integer programming* problems refer to a variant of the above where the variables $\mathbf{x}$ are restricted to be integers. A *mixed-integer optimization* problem comprises both continuous and discrete variables.

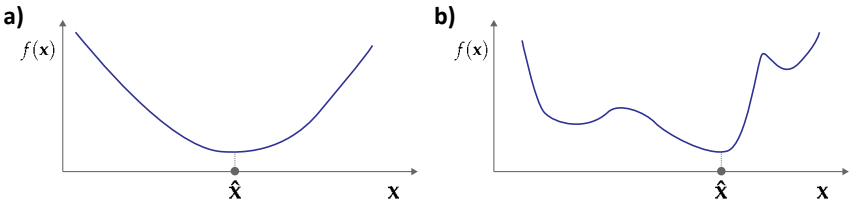


FIGURE 2.9: *Examples of a convex* (a) *and a non-convex* (b) *optimization problem in 1D.*

An important characteristic of an optimization problem is its *convexity*. A convex problem exhibits a convex objective function f and a convex set of feasible solutions Ω (Fig. 2.9, a). This leads to the existence of one global minimum. In contrast, a non-convex problem might comprise many local minima (Fig. 2.9, b) and is therefore generally considered harder to solve.

In this thesis, two optimization algorithms are used: Gradient descent and simulated annealing.

Gradient Descent

Gradient descent is an iterative optimization algorithm that is based on the derivatives of the objective function f. As the gradient of a function points into the direction of steepest ascent, a straightforward method to proceed towards the minimum is to iteratively make steps in the direction of the negative gradient. Starting from an initial position $\mathbf{x}_0$, a sequence of steps

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \nabla f(\mathbf{x}_i)$$

is generated, where $\lambda \in \mathbb{R}_+$ is the step size. For a sufficiently small λ it is guaranteed that $f(\mathbf{x}_{i+1}) \leq f(\mathbf{x}_i)$, leading to a sequence that converges to the closest local minimum (in the case of a convex function this corresponds to the global minimum).

Gradient descent has two main disadvantages: First, the gradient needs to be computed in each iteration. While this is usually not a problem for small to mid-size problem instances, it is a serious concern for objective functions which are defined on large data sets. This is routinely the case for machine learning, where the objective function is defined on an entire training data set, which oftentimes does not even fit into memory. Computing the full gradient for these kinds of problems is computationally infeasible. Second, gradient descent converges to the closest local minimum by construction. For non-convex problems, this can be vastly sub-optimal with high probability, leading to a result that largely depends on the initial position $\mathbf{x}_0$.

An important variation of the basic algorithm is *stochastic gradient descent* (SGD), addressing these two shortcomings. It is concerned with objective functions that comprise a sum of subterms of the form

$$f(\mathbf{x}) = \frac{1}{N} \sum_{k=0}^{N-1} f_k(\mathbf{x}), \tag{2.7}$$

where, typically, each summand f_k evaluates a corresponding data point. In the common *mini-batch*-variant of SGD the gradient of f is approximated by a stochastic

subset of the gradients of its summands:

$$\nabla f(\mathbf{x}) = \nabla \left(\frac{1}{N} \sum_{k=0}^{N-1} f_k(\mathbf{x}) \right) = \frac{1}{N} \sum_{k=0}^{N-1} \nabla f_k(\mathbf{x}) \approx \frac{1}{|S|} \sum_{s \in S} \nabla f_s(\mathbf{x}),$$

where S is a stochastic subset of indices in $[0, N-1]$. In so doing, the gradient computation is approximated by a Monte Carlo estimate and becomes computationally feasible. Furthermore, the thusly injected stochasticity facilitates a random exploration of the domain, which allows the procedure to escape local minima or saddle points.

Simulated Annealing

Simulated annealing is a stochastic optimization algorithm that is often applied in discrete domains, e. g., for integer or mixed-integer problems. It is categorized as a *heuristic*, as it is designed to find a sufficiently good solution without any optimality guarantees, while at the same time being reasonably efficient. The iterative algorithm was inspired by annealing procedures in metallurgy, where a material is subject to heating and controlled cooling to facilitate a re-arrangement of molecules into a lower-energy state.

In each iteration, the algorithm mutates the current state $\mathbf{x}_i$ to a random neighboring state $\mathbf{x}_i^*$. If $f(\mathbf{x}_i^*) \leq f(\mathbf{x}_i)$, the neighbor state $\mathbf{x}_i^*$ is accepted as the new state. If $f(\mathbf{x}_i^*) > f(\mathbf{x}_i)$, the neighbor state is only accepted with probability

$$P(\mathbf{x}_i^*, \mathbf{x}_i, t_i) = e^{\frac{f(\mathbf{x}_i^*) - f(\mathbf{x}_i)}{t_i}},$$

where $t_i \in \mathbb{R}$ is a monotonically decreasing sequence of "temperatures". This construction leads to a random search procedure where new, energy-increasing states are only accepted with a probability that drops with energy difference and over time.

2.6.3 Classical Planning

A classical planning problem is the simplest possible task for an automated planning system. It is concerned with a single agent performing a sequence of deterministic and duration-less actions in a fully observable environment.

Classical planning algorithms are variations of a *state space search*. A state space is a graph (Fig. 2.10, a). Each node V of the graph represents a possible state of the environment and each (possibly directed) edge E corresponds to an action that can be performed to transform one state into another. Searching a state space using *forward chaining* means to start from an initial state V_0 and to successively consider neighboring states with the intention of finding a goal state. *Backward chaining* methods proceed in reverse, starting from one ore multiple goal states. State space search can be either uninformed, i. e., agnostic about the location of the goal state(s), or guided by a heuristic function, that approximates the solution in some way.

Search strategies effectively differ in the order in which states are visited. This thesis makes use of two simple uninformed forward-chaining state space searches: Breadth-first search depth-first search. Both have a time complexity of $O(|V| + |E|)$, i. e., they scale linearly in the size of the graph.

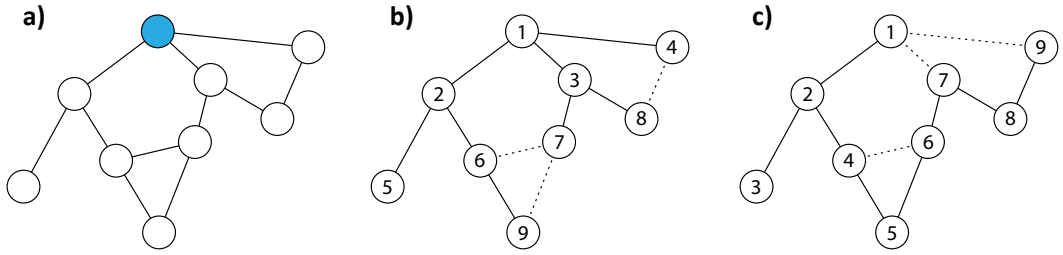


FIGURE 2.10: (a) *A state space with the initial state marked blue.* (b) *Traversal order of a breath-first search.* (c) *Traversal order of a depth-first search. Dotted lines indicate edges to nodes that were explored before.*

Breadth-first Search

Breadth-first search traverses the state space one depth level at a time (Fig. 2.10, b). Starting from the initial state V_0, all neighboring nodes of V_0 are added to a FIFO queue. The algorithm proceeds by sequentially dequeuing nodes, checking them for goal state properties and enqueing respective neighboring nodes that are not yet in the queue. This procedure naturally results in a shortest-path tree.

Depth-first Search

Depth-first search explores the state space as far as possible along each branch (Fig. 2.10, c). Starting form the initial state V_0, all neighboring nodes of V_0 are added to a stack. The algorithm proceeds by sequentially popping nodes V_k from the stack. If V_k has already been explored, it is discarded. Otherwise, V_k is checked for goal state properties and its neighbors are added to the stack.

2.6.4 Machine Learning

Machine learning is concerned with models and algorithms for performing a specific task by relying on inference and pattern recognition from large-scale data. This approach renders explicit, problem-specific instructions largely unnecessary.

The field can be divided into two disciplines, namely *supervised* and *unsupervised* learning. Supervised learning tries to learn a relationship between a space of inputs X and a space of outputs Y from a data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=0}^{N-1}$, with $\mathbf{x}_i \in X$ and $\mathbf{y}_i \in Y$, such that, when given a novel input $\mathbf{x}^*$, its prediction $\mathbf{y}^*$ is accurate. The accuracy is quantified by a *loss function* L, measuring the deviation between predictions and ground truth outputs. Common terms for categorizing supervised learning tasks are *classification* and *regression*, which refer to discrete and continuous output spaces Y, respectively. In contrast, unsupervised learning is concerned with finding a plausible compact representation of the data set by inferring the data generation process to model the data distribution.

In this thesis, two frameworks are used for supervised learning: Linear support-vector machines and deep learning.

Linear Support-vector Machines

A support-vector machine (SVM) is a model for binary classification, i. e., splitting data into two categories. Input data points $\mathbf{x}_i \in \mathbb{R}^n$ are interpreted geometrically as points in n-dimensional Euclidean space. Each point has an associated class label y of

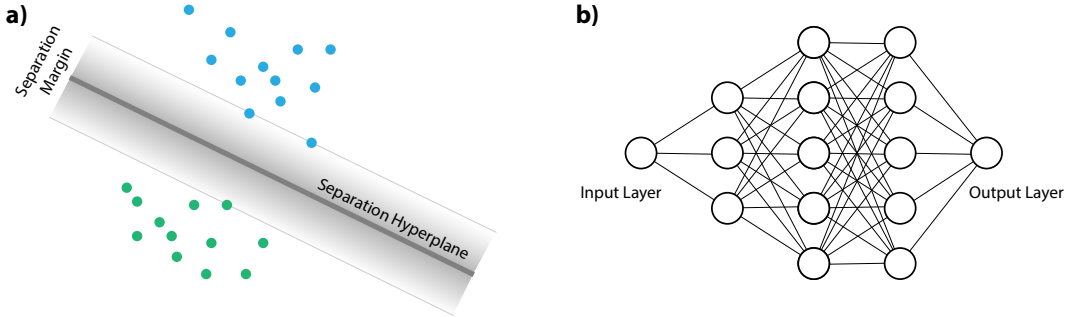


FIGURE 2.11: (a) *A linear SVM classifies the blue and green samples by optimizing for a separation hyperplane surrounded by a margin of maximum size. The shaded areas indicate the value of the hinge loss function (Eq. 2.9, darker means higher value).* (b) *Architecture of a neural network with 5 layers. In this example, every node of each layer receives an input from each node of the previous layer.*

either $+1$ or -1. A linear SVM optimizes for an $(n-1)$-dimensional hyperplane that separates the two classes with the largest separation margin.

A hyperplane can be written using the implicit equation

$$\langle \mathbf{a}, \mathbf{x} \rangle - b = 0, \tag{2.8}$$

where the two parameters we want to find are $\mathbf{a} \in \mathbb{R}^n$, an un-normalized normal vector to the hyperplane, and $b \in \mathbb{R}$, an offset.

SVMs make use of the *hinge loss* function

$$L_i = \max\left(0, 1 - y_i \cdot \left(\langle \mathbf{a}, \mathbf{x}_i \rangle - b\right)\right). \tag{2.9}$$

This loss implicitly defines a margin with size $\frac{2}{\|\mathbf{a}\|_2}$ around the hyperplane of Eq. 2.8 (Fig. 2.11, a). L_i is zero when $y_i \cdot \left(\langle \mathbf{a}, \mathbf{x}_i \rangle - b\right) \geq 1$, i. e., when $\mathbf{x}_i$ lies on the correct side of the margin and therefore delivers the desired prediction y_i. Considering all training data, the complete loss to minimize is

$$L = \left(\frac{1}{N} \sum_{i=0}^{N-1} L_i\right) + \lambda \|\mathbf{a}\|_2, \tag{2.10}$$

where the first term enforces all $\mathbf{x}_i$ to lie on the correct side of the margin, and the second term encourages a large margin width. The hyper-parameter λ determines the trade-off between these potentially conflicting goals. Eq. 2.10 can be optimized by a variant of gradient descent (Sec. 2.6.2), which needs to account for sub-gradients as well, since the loss is not C^1-continuous due to the max function.

Classification of multiple labels is done by a reduction to the binary case. The most simple procedures are to train the SVM to separate one class from all others (*one-versus-all*), or to always consider pairs of classes (*one-versus-one*).

Deep Learning

A learning paradigm that lately received lots of interest is deep learning. It is based on *artificial neural networks*, which is an assemblage of connected units called *neurons*. Each neuron is a non-linear function, accepting possibly multiple inputs and producing a single output. Neurons are cascaded, such that the inputs of a neuron are the outputs of other neurons earlier in the network (Fig. 2.11, b). Commonly, neural networks consist of a multitude layers, forming a *deep* processing structure. Cascades

of non-linear functions are exponentially expressive in the number of layers, which is why deep neural networks are considered universal function approximators [Csáji, 2001]. While neurons can be any non-linear functions, traditionally a neuron consists of a linear combination of inputs followed by a non-linearity called an *activation function*.

Neurons can have trainable parameters referred to as *weights*, which are updated via stochastic gradient descent (Sec. 2.6.2) through reverse-mode automatic differentiation [Linnainmaa, 1970], nowadays commonly referred to as *backpropagation* [Rumelhart et al., 1985]: Given *(i)* a loss function L, *(ii)* a neuron somewhere in the network that computes $y = f(x)$, and *(iii)* the gradient of L with respect to the neuron's output y, the gradient of L with respect to the neuron's input x is computed by applying the chain rule of differentiation,

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial f}{\partial x}.$$

This way, in each step of stochastic gradient descent, the gradient is propagated backwards through the network, eventually updating all trainable weights.

Computer graphics, and in particular filtering recently sees a push towards deep learning-based algorithms. In particular for inverse problems, this idea has led to ground-breaking achievements [Krizhevsky et al., 2012]. Typically, in these convolutional neural networks (*CNN*s), optimization is performed over the space of image convolution kernels.

2.6.5 Probabilistic Reasoning

In many application domains, intelligent systems need to handle uncertainty. This is as environments might not be fully observable and/or deterministic. Furthermore, there might be different options of actions, neither of which has a guarantee of successfully reaching a goal.

Technically, these uncertainties are modeled as random variables. Given the *full joint probability distribution* p of variable assignments, the problem of reasoning under uncertainty reduces to an MLE or MAP estimation problem (Sec. 2.1.5): We want to find the most likely assignment of variables. Representing p is one of the main challenges in the realm of probabilistic reasoning. The number of random variables is usually high, e. g., in computer graphics, each pixel in each frame of a video is routinely treated as a random variable, resulting in hundreds of millions of variables. Therefore, accounting for the joint probabilities of every possible variable assignment naturally results in a combinatorial explosion.

A key technique employed in many probabilistic models is to assume full or conditional independence between certain variables, that is to assume that not every variable influences every other variable *directly*. However, dependency chains might lead to *indirect* influence between variables, retaining global dynamics. Mathematically, independence assumptions lead to a (partially) factorized distribution, which is much more tractable.

Probabilistic graphical models [Koller and Friedman, 2009] are a popular tool for representing (conditional) dependence structures between random variables. The distribution p is modeled as a graph, in which nodes represent the random variables of p and edges indicate dependencies between these variables. The topology of the graph can vary depending on the problem at hand (Fig. 2.12).

In this thesis, a specific kind of probabilistic graphical model is used:

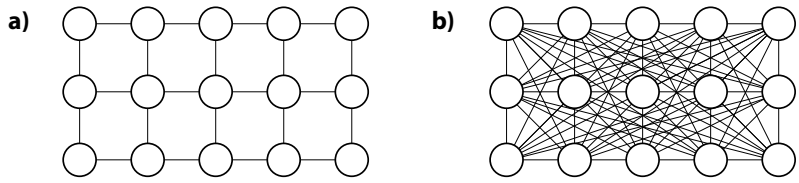


FIGURE 2.12: *Examples of undirected graphical models with variables arranged in a 2D grid structure, as might be used for image pixels.* (a) *A model with adjacency structure, in which only direct neighbors are connected.* (b) *A fully-connected model.*

Conditional Random Fields

Conditional random fields (CRF) [Lafferty et al., 2001] are undirected graphical models encoding relationships between observations $\mathbf{y}$ and random variables $\mathbf{x}$. The probability distribution of the random variables given the observations can be defined as

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{Z(\mathbf{x})} \exp\left(-\left(\sum_i \psi_u(\mathbf{x}_i|\mathbf{y}_i) + \sum_{\mathbf{x}_j \in \mathcal{N}_i} \psi_p(\mathbf{x}_i, \mathbf{x}_j|\mathbf{y}_i, \mathbf{y}_j)\right)\right).$$

Here, $\psi_u(\mathbf{x}_i|\mathbf{y}_i)$ is a unary term, encoding the probability distribution of the random variable $\mathbf{x}_i$ given its corresponding observation $\mathbf{y}_i$, such as for example the probability distribution of depth values of a pixel given that it has a certain color. The pairwise term or edge potential $\psi_p(\mathbf{x}_i, \mathbf{x}_j|\mathbf{y}_i, \mathbf{y}_j)$ encodes how strong two variables $\mathbf{x}_i$ and $\mathbf{x}_j$ influence each other, based on their corresponding observations $\mathbf{y}_i$ and $\mathbf{y}_j$. For example, if two pixels have the same color they are likely to have the same depth. Further, $\mathcal{N}_i$ is the set of random variables that are directly connected to $\mathbf{x}_i$ in the graphical model. Finally, the normalizing partition function Z ensures that p is a valid probability distribution.

MAP inference in conditional random fields of relevant size is considered difficult and many algorithms have been designed for solving this problem exactly or approximately [Koller and Friedman, 2009]. Notably, Krähenbühl and Koltun [2011] have shown how inference in fully-connected CRFs with Gaussian edge potentials can be performed by obtaining a mean-field approximation of the probability distribution using bilateral filtering [Tomasi and Manduchi, 1998].

Chapter 3

Deep Point Correlation Design

3.1 Introduction

Point patterns have many important uses in computer graphics, linking apparently disparate topics such as natural placement of procedural plants, accurately casting shadows from an area light or placing artistic stipples in a visually pleasing fashion. Many classic algorithms have been proposed to generate point patterns e. g., [Lloyd, 1982] relaxation algorithm or dart throwing [McCool and Fiume, 1992]. Their properties are analyzed in terms of spectra [Yellott, 1983], differentials [Bowers et al., 2010; Wei and Wang, 2011] and their application as samples in Monte Carlo (MC) integration [Cook, 1986; Subr and Kautz, 2013; Pilleboue et al., 2015; Singh and Jarosz, 2017].

Designing methods to provide the desired pattern quality for different applications is an active topic of research [Wei and Wang, 2011; Fattal, 2011; Zhou et al., 2012; De Goes et al., 2012; Heck et al., 2013; Kailkhura et al., 2016]. Devising such point patterns typically requires complex mathematical derivations which are only applicable in very specific conditions, implementation effort, and finally compute time in order to run an optimization which produces a point set.

In this chapter we add a new level of abstraction and suggest to use modern deep learning to optimize over the space of point pattern generation methods itself. Instead of mathematical derivation, a user of our system provides a straightforward implementation of the desired properties in form of an *agenda* snippet demanding e. g., "a blue noise spectrum both in 2D but also projected to the x axis". We then optimize over the learnable parameters of a *deep generative model* consisting of weighted distance-based unstructured filters of compact support that map random point sets to point patterns. Notably, we learn this in a weakly supervised fashion, without observing any instances of the desired point patterns – some of the patterns we produce are not known how to produce – but directly from a description of the desired properties alone. The pipeline is deep, so that several steps of correction (typical ca. 40) can occur and communicate. We use filters based on distance measured in all subspaces which allows both scaling to high dimensions (separability) but also supporting anisotropy that can achieve different characteristics in different subspaces. Our unstructured filters support sparse point sets, as a dense representation would not scale to high dimensions. After the optimization i. e., learning, has finished, producing points does only require running the resulting recursive GPU filters and no optimization.

Rendering is a key application of point patterns, where deep learning has been used for relighting [Ren et al., 2015], screen-space shading [Nalbach et al., 2017], volume rendering [Kallweit et al., 2017] or de-noising [Chaitanya et al., 2017], light transport [Dahm and Keller, 2017] as well as importance sampling [Müller et al., 2018; Zheng and Zwicker, 2018]. While Müller et al. [2018] and Zheng and Zwicker [2018] also address sampling, their work is focused on building deep models of the indirect

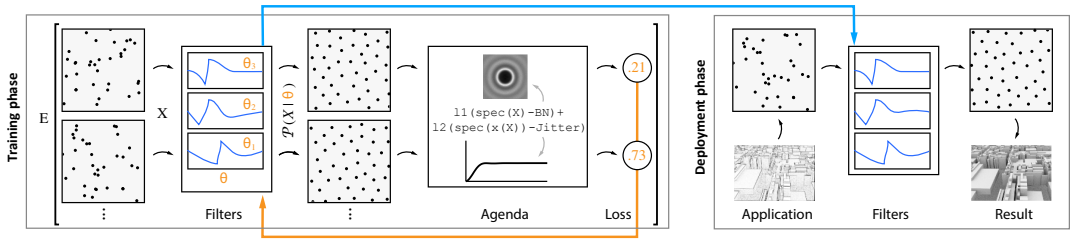


FIGURE 3.1: *Overview of our system, comprising of two main parts: learning* (left) *and deployment* (right).

light field function in a specific scene and placing samples where this function is high in an importance-sampling spirit. Note, that this is a scene-dependent process, that is trained per-scene, while our approach is trained once to find point patterns that generalize across all scenes / integrands.

We make use of learned operations on point clouds, as pioneered by PointNet [Qi et al., 2017], but using non-linear filters, that generalize unstructured linear convolution [Hermosilla et al., 2018]. Instead of inferring labels or per-pixel or per-point attributes such as normals, we optimize for filters that transform sets of random points into sets of points with the desired properties.

Our system introduces a point pattern *agenda*, a notation to define point pattern requirements using programmatic expressions. This is a simple instance of a domain specific language, such as recently proposed for image synthesis [Anderson et al., 2017], non-linear image optimization [Devito et al., 2017; Heide et al., 2016] or physics [Bernstein et al., 2016]. Instead of deriving our own parser, we provide functions in TensorFlow [Abadi et al., 2016] that are parsed and evaluated efficiently during training and testing using TensorFlow's symbolic analysis and GPU evaluation support.

In summary the contributions made in this chapter are: *(i)* A GPU-friendly method to generate point patterns using recursive weighted distance-based unstructured filtering in high dimensions, *(ii)* a method to learn these filters from prescribed design goals alone, without math or coding, *(iii)* novel point patterns such as high-dimensional isotropic BN, and mixed forms of BN in high dimensions and subspaces, and *(iv)* an unbiased MC estimation of the radially-averaged spectra that allows for analysis and optimization in high dimensions.

3.2 Overview

Our exposition has two main parts (Fig. 3.1): First, we introduce the notion of a point correlation design agenda (Sec. 3.3) which defines the desired properties. Second, we describe a deep architecture (Sec. 3.4) that can be optimized in respect to this design agenda.

An *agenda* is a functional programming snippet, that maps a point pattern to a scalar value. It is the only user-provided input to our system at training time. Devising operations to compose agendas which allow defining point patterns is the first key technical contribution of this chapter. We optimize for a point generation method to realize the agenda by back-propagating it through a neural network architecture. This architecture is our second main contribution and consists of a deep cascade of weighted distance-based, unstructured filters (Sec. 3.4). The unstructured representation we choose allows to scale to point sets that are naturally sparse in high dimensions (>3D) and could not be captured using regular representations commonly

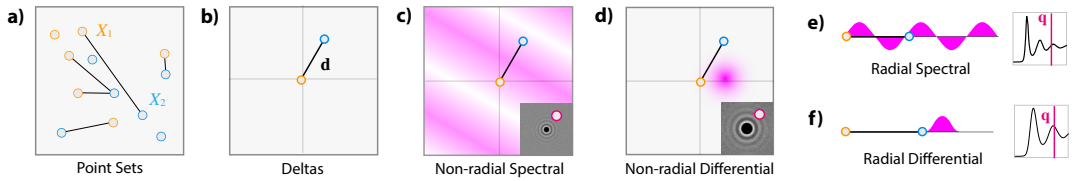


FIGURE 3.2: *Point Correlation (Eq. 3.1), here shown for a 2D example, between an orange point set X_1 and a blue point set X_2 is defined as an expected value over all pairs of points from the respective sets* (a). *We here show five point pair examples. For each pair the deltas are produced by subtracting the first point* (b), *here shown for a single pair. This delta is then given to a function κ that can work in different ways, out of which we illustrate four* (c-f). c) *and* d) *work in 2D and correlate with the 2D Fourier basis* (c) *or the Gaussian basis* (d) *at all 2D correlation coords* $\mathbf{q}$. e) *and* f) *proceed the same, just that the correlation coordinate is a single scalar distance* $\mathbf{q}$.

used for 2D images or 3D volumes. Our filters map high-D signals to high-D signals, but rely on weighted distances only. This avoids the curse of dimensionality that common filter masks encounter: While, typically, high-dimensional filters depend exponentially on the number of dimensions, our filters scale linearly. In a SIMD compute model they are even constant. Finally, our filters have compact support, allowing to execute fast, once trained, for large point numbers and high dimensions.

Our system further is comprised of two stages (left and right in Fig. 3.1): a learning stage in which the architecture is optimized to fit the agenda and a deployment part, in which the result of this optimization is executed to generate new point patterns. The training is supervised only by the agenda, that measures point pattern quality and never requires supervision by any example results of any point pattern generation. Therefore, our supervision strategy can be classified as *inexact* [Zhou, 2017]. In particular, our approach produces patterns which were previously unknown and consequently could not have been input to a traditional supervised learning.

3.3 Point Pattern Agendas

Inspired from design theory we call our losses *agendas* to emphasize that they are not hard-wired parts of our system (which a loss typically is), but user input. We define a range of point correlation operators that form a simple domain-specific language for point pattern design. We will start by simple spectral and differential domain properties, include linear and non-linear projections, introduce coverage and finally discuss the metrics between point correlations.

Note, that the key idea to allow our method to become scalable is, that all point correlation operators are only ever computed at training time, never at test time. Training is slow; execution is fast.

3.3.1 Notation

An agenda $a(X) \in \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{+}$ maps an unstructured set X of m points in n-dimensional space to a single non-negative scalar, which is smaller for point patterns that are closer to the agenda's goal. In the notion of deep learning, the agenda implements a loss. The agenda can make use of linear combinations of terms comprising custom operations to be listed next. When combining multiple terms, it is still the end-user's responsibility to scale them appropriately, such that they fall in a compatible range.

3.3.2 Point Correlation

To simplify the exposition, we will define the point set correlation (Fig. 3.2) between point set X_1 and point set X_2 as

$$P_\kappa(X_1, X_2)(\mathbf{q}) = \mathrm{E}_{\mathbf{x}_1 \sim X_1}[\mathrm{E}_{\mathbf{x}_2 \sim X_2}[\kappa(\mathbf{x}_1 \ominus \mathbf{x}_2, \mathbf{q})]], \tag{3.1}$$

where $\mathrm{E}_{a \sim A}$ is the expected value of the random variable A, κ is a kernel, $\ominus$ is the toroidal vector difference and $\mathbf{q}$ is what we will call the *correlation* coordinate.

The notion of P_κ is a two-fold abstraction. It generalizes along one axis over Fourier spectra [Zhou et al., 2012], the differential domain [Wei and Wang, 2011] but also across properties such as saturation [Mitchell et al., 2018] defined on pairs of point sets. Along a second axis, generalization is folded into κ, which can work both on linear high-D offset and on non-linear 1D distance. $P_\kappa(X_1, X_2)(\mathbf{q})$ is a distribution across the correlation coordinate $\mathbf{q}$, which is again an abstraction of frequency or distance of points. The abstract notion is implemented by choosing a correlation kernel κ. The kernel puts $\mathbf{x}_1 \ominus \mathbf{x}_2$ and $\mathbf{q}$ in different relations: it can work on distances or offsets as well as it can apply non-linearities such as complex exponentiation followed by a norm.

We will frequently use $X_1 = X_2$, the correlation of a point set with itself. If we analyze a point pattern in respect to itself, we will shorthand write $P(X)$ in place of $P(X, X)$.

As $P_\kappa(X_1, X_2)(\mathbf{q})$ is an expected value, it could be evaluated using quadrature or estimated using Monte Carlo. We will however use specifically optimized implementations for specific κ such as Fourier or the differential domain [Wei and Wang, 2011].

Eq. 3.1 is similar to Eq. 6 in Wei and Wang [2011], but ours is using toroidal distances, two point sets instead of one and formulated using expected values instead of integrals to match the way they will be evaluated in back-propagation.

3.3.3 Spectrum

The spectrum of a point set X is computed using

$$\mathtt{spec}(X) = P(X) \qquad \text{with} \qquad \kappa(\mathbf{d}, \mathbf{q}) = \cos(2\pi \langle \mathbf{d}, \mathbf{q} \rangle)$$

and maps the n-dimensional sparse point pattern to an n-dimensional spectrum. In this case $\mathbf{q}$ is a n-dimensional frequency. The resolution n_c of the spectrum can be configured by the user, but is typically chosen to be a multiple of $m^{1/n}$.

This operation is commonly used in conjuction with a norm such as $\mathcal{L}_1$ to measure the difference to a reference spectrum. The agenda

$$\alpha(X) = \mathtt{l1}(\mathtt{spec}(X), \mathtt{BNOT})$$

for example computes the spectrum of a pattern X and compares it to a reference blue noise spectrum `BNOT`. As the DC term (frequency 0) amounts to be the number of points, and we would like to train independently of that number, we decided to remove it from the spectrum i. e., set it to zero as in $P(X)(0) := 0$.

While P is in general defined on pairs of points in Eq. 3.1, it can be estimated by a single loop over all points (Eq. 2.5).

3.3.4 Differential Domain

The function

$$\texttt{dDom}(X) = P(X) \qquad \text{with} \qquad \kappa(\mathbf{d}, \mathbf{q}) = \mathcal{N}(\mathbf{q} - \mathbf{d})$$

maps the sparse point pattern to a dense distribution of offsets, where $\mathcal{N}$ is a zero-mean Gaussian with a standard deviation of $2/n_c$. Effectively, a n_c-bin n-D histogram. The number of bins could be chosen optimally in respect to spectrum and point count [Scott, 1979], but we found an empirically chosen value of 128 to perform well across all experiments. In this case $\mathbf{q}$ is a n-dimensional differential coordinate. It is to be used similar to the spectrum e. g.,

$$\alpha(X) = \texttt{l2}(\texttt{dDom}, \texttt{Jitter})$$

would ask for a power histogram that is $\mathcal{L}_2$-similar to the power histogram `Jitter` of jittered sampling.

For computation, we iterate all pairs of points, compute their distance, and scatter them to the respective bins around $\mathbf{q}$ with Gaussian weights instead of iterating all pairs for all values of $\mathbf{q}$. Linear complexity can be achieved by only iterating over points within a certain distance range [Wei and Wang, 2011]. Note how the Gaussian weights make this process back-propagatable.

3.3.5 Radial Mean

Besides producing an n-D spectrum or histogram from n-D points we also support working on the radially-averaged spectrum

$$\texttt{radSpec}(X)(d) = \mathrm{E}_{\mathbf{q}\sim\Omega}\, \texttt{spec}(X)(d \cdot \mathbf{q})$$

and the radially-averaged histogram

$$\texttt{radDDom}(X)(d) = \mathrm{E}_{\mathbf{q}\sim\Omega}\, \texttt{dDom}(X)(d \cdot \mathbf{q}),$$

where d is a radius and Ω is the n-dimensional unit hyper-sphere.

Monte Carlo estimate A trivial construction of `radSpec` would not operate on the point set X, but on $\texttt{spec}(X)$. This indeed would be more modular, but regrettably does not scale well to high dimensions like $n = 10$, where a regularly sampled spectrum would require prohibitive amounts in the order of $\mathcal{O}(n_c^n)$ of memory. Note, that the output of the radially averaged spectrum is always a compact 1D function of radius, that has $\mathcal{O}(1)$ memory requirements.

Addressing this difficulty, we suggest estimating the spectrum using Monte Carlo integration as follows: First, we generate random points $\mathbf{q}$ on the hypersphere Ω using the method of Hicks and Wheeling [1959] and scale them by d. Since we are only interested in integer frequencies, we then snap the points to the integer grid. Finally, we evaluate $P(d \cdot \mathbf{q})$ as described above and average. For low-dimensional point correlations (in practice $n \leq 2$), we evaluate all $\mathbf{q}$ exhaustively in a grid using quadrature.

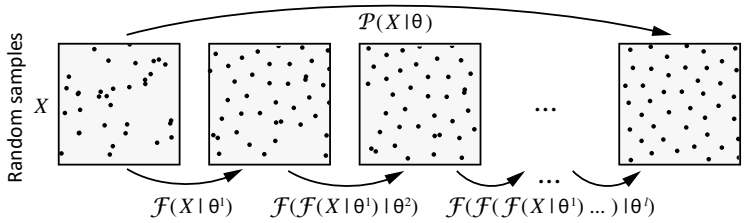


FIGURE 3.3: *Notation example for $m = 32$ in $n = 2$, showing, from left to right, the first two filters and the final result.*

3.3.6 Anisotropy

When computing a radial spectrum, we average across points on hyperspheres. In the same way as a mean, we can compute the variance of those distributions. Same as the radial average, this is a mapping from an n-D point set to a 1-D distribution of variance across distances, defined as

$$\texttt{radVarSpec}(X)(d) = \mathrm{E}_{\mathbf{q}\sim\Omega}\left(\texttt{radSpec}(X)(d) - \texttt{spec}(X)(d\cdot\mathbf{q})\right)^2$$

and analogous for the differential domain. Again both can be solved using quadrature or MC integration.

3.3.7 Swizzle

As we are interested in the properties of subspaces, we make use of the typical swizzle operations denoted as $\texttt{x}(X)$ for the x component, $\texttt{y}(X)$ for the y component, $\texttt{x}y(X)$ for the xy component, etc.

3.3.8 Metrics

Typical metrics to compare n-D spectra and histograms (distributions) are $\mathcal{L}_1$, $\mathcal{L}_2$, which we denote as `l1`, `l2`. In our experiments, we mostly use $\mathcal{L}_1$ thanks to its robustness. It is commonly achievable in deep learning, but harder to achieve in mathematical derivations, that often revert to $\mathcal{L}_2$.

3.4 Point Patterns via Iterated Filtering

Here we introduce a deep neural network generating point patterns. The idea is to use a deep pipeline (Sec. 3.4.1) of learnable filters (Sec. 3.4.2) that convert a high number of random high-D points into points that follow the agenda formalism just introduced in Sec. 3.3.

3.4.1 Architecture

We formalize this as learnable mapping $\mathcal{P}_\alpha(X|\Theta)$ from a set of m uniformly random n-D input points X_{In} without any specific properties, into a set of m output points $X_{\text{Out}} = \mathcal{P}(X_{\text{In}}|\Theta)$ with specific properties according to the agenda α. The mapping $\mathcal{P}$ has tunable parameters Θ consisting of filter weights $\mathbf{w}$ (Sec. 3.4.2, Distance-based

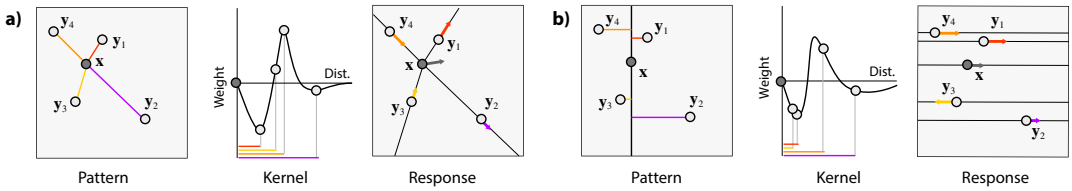


FIGURE 3.4: *Our tunable filters, based on distance* (a) *and based on weighted distances* (b). *The "Pattern" images show a point* **x** *in 2D and all other pattern points. Colored lines indicate difference between other points and the center point. The "Kernel" images show the filter kernel (x axis is distance, y axis is weight). The four color-coded distances are used to look up the kernel weight. The "Response" images show how the offset vectors between* **x** *and all other points are scaled by the kernel response, and added, resulting in a single dark-grey correction vector that moves* **x**. a) *uses common distances,* b) *uses weighted distance (here x-only).*

Filters) and combination matrices S (Sec. 3.4.2, Combination), for which we SGD-optimize i. e., learn,

$$\Theta = \arg\min_{\Theta'} \mathrm{E}_{X\sim\mathcal{U}^{m\times n}}[\alpha(\mathcal{P}(X|\Theta'))].$$

We choose to implement $\mathcal{P}$ as a cascade

$$\mathcal{P} = \mathcal{F}(\mathcal{F}(\ldots\mathcal{F}(X|\theta^{(1)})\ldots|\theta^{(l-1)})|\theta^{(l)})$$

of l unstructured, weighted-distance filters $\mathcal{F}$ with learnable parameters

$$\Theta = \{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(l)}\}.$$

These filters simply map a point set of a certain size and dimension to another point set of the same size and dimensions and will be described next.

3.4.2 Filters

Our filters are required to work on unstructured data, i. e., a list of n-D points. PointNet [Qi et al., 2017] and following papers have made use of symmetric functions and rotational transformers in a tunable fully-connected architecture, but without translation-invariance, i. e., using fully connected settings. This might work when sampling a chair into a small number of points in 3D, but not for many points in high dimensions. Other work has generalized image convolutions to unstructured 3D by phrasing the convolution kernel as a neural network [Hermosilla et al., 2018], but without scalability to higher-dimensional convolution domains. Our filters are both convolutional and scale to high dimensions. We will first introduce distance-based filters (Sec. 3.4.2), limited to isotropic agendas, before we generalize them to weighted-distance based filters (Sec. 3.4.2) that can further be used for anisotropic designs.

Distance-based Filters

A first solution is to work on one-dimensional distances, i. e., with radially symmetric kernels, as shown in Fig. 3.4, a. These are simple enough to be represented using a 1D table $\mathbf{w}$ of weights with b entries $\mathbf{w}_i$. As the computational complexity of our filters is independent of the table size, we empirically set $b = 128$ in all our experiments.

To avoid computing interaction between all points, which would imply quadratic time complexity, we limit the filters to a constantly-sized neighborhood of a *receptive field* r, that is typically chosen to be only a fraction of the domain.

Reading the kernel table continuously at distance $d \in (0, r)$ using linear sampling, makes the response differentiable, and therefore back-propagatable. We also always learn a residual, i. e., an offset to adjust the point position, which improves the gradient flow.

Let a kernel g, parametrized by a b-D weight vector $\mathbf{w}$ be

$$g(d|\mathbf{w}) = (1 - \text{frac}(\hat{d})) \cdot \mathbf{w}_{\lfloor \hat{d} \rfloor} + \text{frac}(\hat{d}) \cdot \mathbf{w}_{\lceil \hat{d} \rceil} \quad \text{where} \quad \hat{d} = d \cdot r / b.$$

Now filtering $\mathcal{F}'$ with the kernel g is defined as

$$\mathcal{F}'(\mathbf{x}|\mathbf{w})_i = \mathbf{x}_i + \sum_{j \neq i}^{m} g(||\mathbf{x}_j \ominus \mathbf{x}_i||_2 | \mathbf{w}) \frac{\mathbf{x}_j \ominus \mathbf{x}_i}{||\mathbf{x}_j \ominus \mathbf{x}_i||_2}, \tag{3.2}$$

where $\ominus$ denotes the toroidal vector difference. In a slight abuse of notation, we will refer to the (overloaded) filtering of all points X as $\mathcal{F}(X)$ as well. See the appendix for a derivation of the gradients required for more efficient learning.

Note, that the weights $\mathbf{w}$ can – and also need to be – negative. Positive weights attract $\mathbf{x}_i$ into the direction of $\mathbf{x}_j$, negative weights repel. This is shown by the positive and negative weights in the column "Kernel" in Fig. 3.4, a.

Note that the number of filter kernel bins only adds to the number of trainable parameters, but does not affect the speed at deployment time: It is a $O(1)$ table look-up.

This formulation is similar to the one of Zhou et al. [2012] and Heck et al. [2013], who also update point positions with new positions in an optimization. The "Response" column in Fig. 3.4, a visualizes this: some arrows (orange, yellow) point towards $\mathbf{x}$, the others (red, violet) point away. The sum of all these is applied to $\mathbf{x}$ (black arrow). However, we do not perform a costly optimization for a given point set X using analytic derivations at run-time, but instead optimize over all methods that are suitable for fast execution of GPUs. Furthermore, our notion generalizes to subspaces as we will explain in Sec. 3.4.2.

Gradients

Our unstructured filters (Eq. 3.2) can be used directly in a trivial TensorFlow implementation. However, we found this to execute much slower than a hand-crafted C++ implementation and most of all required memory in the order of $O(m^2)$. We here give the derivation required to implement our filter as a custom operation: a forward pass to filter points ($\mathcal{F}$, Eq. 3.2) and the derivative passes in respect to kernel weights parameters ($\partial L / \partial \mathbf{w}$) and input point positions ($\partial L / \partial \mathbf{x}$).

Both the *forward pass* and the *backward pass* are implemented parallel over all points $\mathbf{x}_i$. Each point sequentially iterates all neighbors $\mathbf{x}_{i,j}$. Distances d_{ij} are computed and the weights $\mathbf{w}$ are indexed at $\lceil d_{ij} \rceil$ and $\lfloor d_{ij} \rfloor$, interpolated, and applied to the deltas $\mathbf{x}_i \ominus \mathbf{x}_j$.

The *backward pass* requires the gradient of the loss L of Eq. 3.2 in respect to the l-th weight $\mathbf{w}$, which is

$$\frac{\partial L}{\partial \mathbf{w}_l} = \sum_i \sum_{j \neq i} \frac{\partial g(d_{ji}|\mathbf{w})}{\partial \mathbf{w}_l} \frac{1}{d_{ij}} \left\langle \mathbf{x}_i \ominus \mathbf{x}_j, \frac{\partial L}{\partial (\mathcal{F}(\mathbf{x}|\mathbf{w}))_i} \right\rangle,$$

a b-dimensional vector, where $d_{ab} = ||\mathbf{x}_a \ominus \mathbf{x}_b||$. Here,

$$\frac{\partial}{\partial \mathbf{w}_l} g(d|\mathbf{w}) = \begin{cases} 1 - \text{frac}(\hat{d}) & \text{if } \lfloor \hat{d} \rfloor = l \\ \text{frac}(\hat{d}) & \text{if } \lceil \hat{d} \rceil = l \\ 0 & \text{else} \end{cases},$$

where $\hat{d} = d \cdot b/r$ is the index scaled by receptive field and bin count and $\text{frac}(\hat{d})$ returns the fractional part of a real $\hat{d}$. The derivative of Eq. 3.2 in respect to the k-th dimension of the i-th input point is

$$\frac{\partial L}{\partial \mathbf{x}_{i,k}} = \frac{\partial L}{(\partial \mathcal{F}(\mathbf{x}|\mathbf{w}))_{i,k}} + \left\langle \sum_{j \neq i} \frac{\partial \otimes_{ji}}{\partial \mathbf{x}_{i,k}}, \frac{\partial L}{(\partial \mathcal{F}(\mathbf{x}|\mathbf{w}))_i} \right\rangle + \sum_{j \neq i} \left\langle \frac{\partial \otimes_{ij}}{\partial \mathbf{x}_{i,k}}, \frac{\partial L}{(\partial \mathcal{F}(\mathbf{x}|\mathbf{w}))_j} \right\rangle,$$

where $\otimes_{ba}$ is the pairwise interaction between point b and a

$$\otimes_{ba} = g(||\mathbf{x}_b \ominus \mathbf{x}_a||_2|\mathbf{w}) \frac{\mathbf{x}_b \ominus \mathbf{x}_a}{||\mathbf{x}_b \ominus \mathbf{x}_a||}$$

that has the derivative

$$\frac{\partial \otimes_{ba}}{\partial \mathbf{x}_{b,k}} = \frac{1}{d_{ba}^2} \Big(\frac{1-b}{r} (w_{\lceil \hat{d} \rceil} - w_{\lfloor \hat{d} \rfloor})(\mathbf{x}_{b,k} \ominus \mathbf{x}_{a,k})(\mathbf{x}_b \ominus \mathbf{x}_a) + g(d_{ba}|\mathbf{w}) \cdot \Big(\frac{(\mathbf{x}_{b,k} \ominus \mathbf{x}_{a,k})(\mathbf{x}_b \ominus \mathbf{x}_a)}{d_{ba}} - \mathbb{1}_k d_{ba} \Big) \Big),$$

where $\mathbb{1}_k$ is a one-hot vector. Finally,

$$\frac{\partial \otimes_{ba}}{\partial \mathbf{x}_{a,k}} = -\frac{\partial \otimes_{ba}}{\partial \mathbf{x}_{b,k}}.$$

Weighted Distance-based Filters

We found the distance-based unstructured filters to work well, if the agenda does not ask for anisotropic effects: When requiring an elliptical spectrum, a filter based only on distances has no way of disambiguating if a distance of .1 was along the x or y direction, where it means something different. In other words the agenda can ask for more than what the architecture can achieve.

As a solution, it is tempting to use full-dimensional filters, but a kernel would now need to learn up to all, say, 10-D interactions, resulting in b^{10} learnable parameters, so at least $3^{10} = 49,304$ parameters for each filter, i. e., way too much to learn or even execute efficiently.

As a middle ground, we opt to learn combinations of separable filters. The weighted distances between two points $\mathbf{x}_1$ and $\mathbf{x}_2$ is $||\mathsf{M}(\mathbf{x}_1 \ominus \mathbf{x}_2)||_2 \in R^n \to \mathbb{R}^+$ where M is a (diagonal) weight matrix. The common distance is a special case of an identity weighting $\mathsf{M} = \mathbf{1}$. The weighted distance-based filter is now

$$\mathcal{F}(\mathbf{x}|\mathbf{w})_i = \mathbf{x}_i + \sum_{j \neq i}^{m} g(||\mathsf{M}(\mathbf{x}_j \ominus \mathbf{x}_i)||_2|\mathbf{w}) \frac{\mathsf{M}(\mathbf{x}_j \ominus \mathbf{x}_i)}{||\mathsf{M}(\mathbf{x}_j \ominus \mathbf{x}_i)||_2}. \tag{3.3}$$

By choosing different weight matrices M, different subspaces can be addressed individually, respectively others can be ignored, by setting a column to zero. In practice,

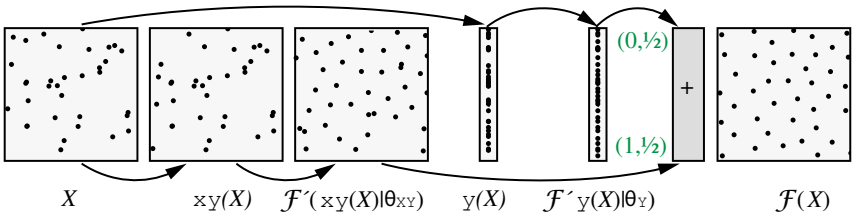


FIGURE 3.5: *Combination example for an x, xy agenda.*

we only use such M that are either 0 or 1 on the diagonal and encode this by simply listing the non-zero elements, as done in the swizzle operation.

Please note that we learn the filters of the kernels $\mathbf{w}$, while the weight matrix M is input to our method. The weight matrix is found automatically from the agenda: whenever the agenda asks for a property in any specific subspace (swizzling), a special convolution for this subspace is created.

Combination

The weighted distance-based filtering can learn how to apply filters in subspaces, but to be useful, multiple subspaces (e. g., x and y) need to work together, as well as jointly with the space they are a subspace of (xy). To this end, the outcome of multiple subspaces needs to be combined. In particular, this combination has to be done after each layer. The necessity to do so is seen when considering an agenda optimizing x and xy jointly (Fig. 3.5): after each adjustment in each dimension, both outcomes affect the other; they need to share information and have to produce one joint xy result, not two disparate x and xy results.

We achieve this by a trainable normalized sum of all subspaces. Its trainable parameters form a real matrix $\mathsf{S} \in \mathbb{R}^{n \times s}$, that is multiplied with the stacked vector of all output coordinates from all subspaces. The normalization divides each element in that vector by the number of occurrences in all subspaces. An agenda that e. g., asks for x, xy and xyz would add the three, but divide x by 3 and y by 2 (green numbers in Fig. 3.5). This normalization allows initializing the weights in subspaces without considering how they are to be used later.

Gridding

Our method can operate in a gridded and ungridded mode. The ungridded is the default and used for general point patterns. The gridded variant holds some dimensions fixed.

In a *non-gridded* point pattern, all dimensions are filtered as processed and $\mathcal{P}$ maps from n to n dimensions. A point pattern is *gridded* when it maps from n_{In} input to n_{Out} output dimensions. Here, the first $n_{\text{In}} - n_{\text{out}}$ dimensions are initialized using a regular grid. These values are left unchanged by the filter pipeline. However, all n_{In} values are used to compute distances.

An example of a gridded point pattern with $n_{\text{In}} = 3, n_{\text{Out}} = 1$ is a pattern where the first two fixed dimensions are the pixel centers, and the third dimension is the wavelength for spectral rendering. In other words, a gridded pattern is a height field: x and y are implicit and fixed, and the height z changes. Our method can now learn a $\mathcal{P}$ that arranges the wavelength such that they are BN in respect to other nearby wavelengths at fixed spatial positions.

3.4.3 Training

Variance Reduction Individual patterns give rise to noisy point correlations, which is why the set correlations of multiple realizations are typically averaged, even for visualization. In particular for our Fourier design, we found the use of this unbiased estimate essential for convergence: In each training iteration, multiple output spectra of our system are averaged before they are evaluated within the agenda, e. g., by comparing them to a target. This stabilizes training significantly, especially for small point counts m. Crucially, the agenda designer does not have the freedom to combine spectra of different realizations arbitrarily: Spectra are always averaged. Please notice that this procedure is subtly different from mini-batching, as the latter requires a loss function that linearly adds terms corresponding to the training data points (Eq. 2.7). In a slight abuse of notation, we nevertheless refer to the point set realizations used for variance reduction as a batch. We observe that a batch size of 2 samples – followed by radial averaging – leads to spectra that are converged enough to be used for gradient computations.

Optimization Training is implemented in TensorFlow using the ADAM optimizer, with an exponentially decreasing learning rate initialized by 10^{-6}. Learning typically requires around 8 hours on an Nvidia Tesla V100-PCIE with 32GB memory and 1.38GHz memory clock rate.

3.4.4 Discussion

The trained network $\mathcal{P}$ is specific to the dimension n and the number of points m. While the filters work on distances, and are agnostic to dimensions, what they learned is a dimension-dependent task. Effectively, the network needs to be re-trained for every dimension.

Our architecture is trained for a fixed number of points m. While the network can also be applied to similar numbers of points, drastically different point numbers require re-training.

A user also has to choose the depth of the network l and the trained filters will be specific to that depth. As we find the solution to be progressive, i. e., the quality to improve with layers, it is possible to stop computation earlier, at $l' < l$, but the maximum quality is achieved at l. One solution is to train with a high l and then cut compute time at deployment to an l' with the required quality.

In particular, we found the filter parameters θ to not be the same at different depths, so $\theta^{(a)} \neq \theta^{(b)}$ in general. We also tested, how training "siamese", with all θ in lock-step, decreases quality.

Applying the cascade of filters is similar to an update in a Jacobi or Gauss-Seidel-type optimization or a generalized step of Lloyd relaxation but with learned update rules. In this light, our approach uses non-linear filters to predict gradients in an unrolled gradient-descent type optimization.

The filters we employ are non-linear by construction. We also experimented with linear filters and introducing explicit non-linearities such as ReLUs but did not observe an improvement. Also note, that, subtle enough, using a linear filter would imply linear offsets, and these are subject to the curse of dimensionality.

As our training set comprises uniform random vectors, the concept of epochs that relates to finite training examples is not applicable. Instead, we will refer to learning effort in units of batch counts, i. e., how many pattern points were produced, divided by the number of patterns per batch. The number of point sets per batch needs to be

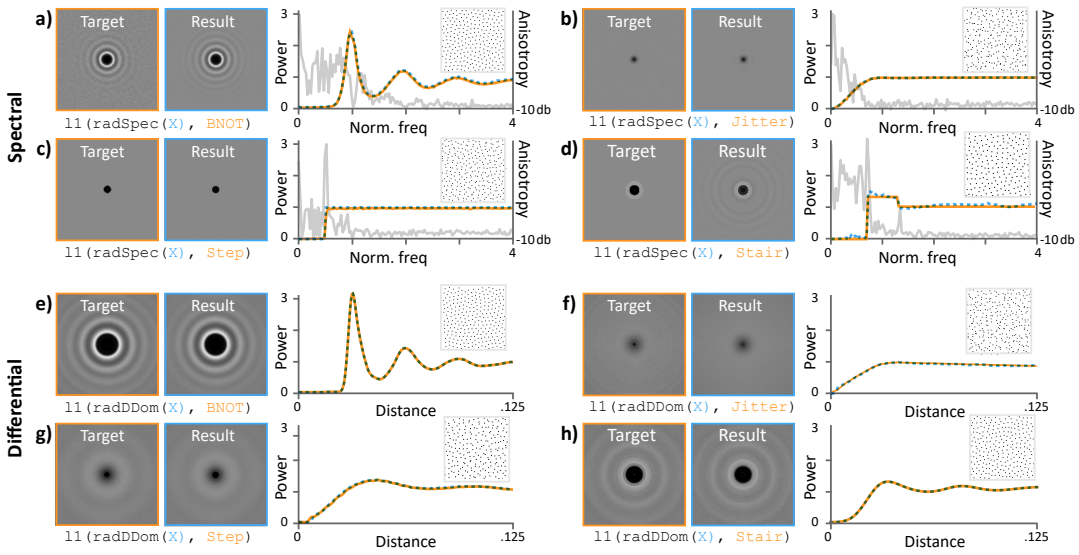


FIGURE 3.6: *Results for isotropic 2D agenda for different isotropic spectral* (a-d) *and differential domain targets* (e-h). *Each block shows the source and target 2D spectrum / PCF, the source (dotted blue) and target (orange) radial average the anisotropy (grey, only for Fourier, vertical scale different from power).*

larger than one for effective training to remove the realization noise. Test and train split is realized by using different random seeds to generate the input patterns.

3.5 Results

Here we perform a quantitative analysis for our approach in terms of spectral and differential properties (Sec. 3.5.1), we look into Monte Carlo convergence and discrepancy (Sec. 3.5.2), we instrument different aspects of scalability and parameter choices (Sec. 3.5.3) and finally show applications to rendering and object placement (Sec. 3.5.4).

Default Parameters Unless said otherwise, this section uses $m = 1024$ points, $n_c = 128$ pixels / bins for full-dimensional and, $n_a = 4m^{\frac{1}{n}}$ for radially averaged spectra or histograms, networks with a depth of $l = 45$ layers / filters, where each filter has a kernel with $b = 96$ elements which span a receptive field of $r = .5$.

Presentation Spectra or histograms shown are averaged across 1000 realizations. Our radially averaged spectral plots are scaled horizontally by $m^{-1/n}$ to support comparison across different numbers of points m and dimensions n and cropped to the range $[0, 4]$. We call this *normalized* frequency.

3.5.1 Spectral and Differential Analysis

Isotropic 2D

We start by showing results from learning several state-of-the-art isotropic 2D patterns in Fig. 3.6. All agendas minimize an `l1` difference to radially averaged reference spectra.

We see, that our approach can produce four relevant methods (BNOT [De Goes et al., 2012], jitter [Cook, 1986], Step BN [Heck et al., 2013] and Stair BN [Kailkhura

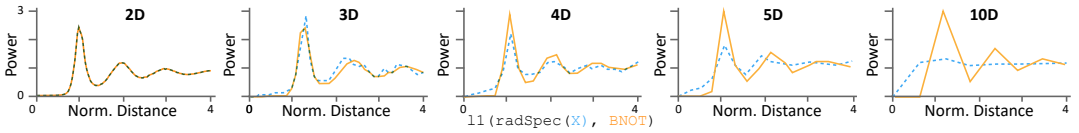


FIGURE 3.7: *Results for isotropic agenda in higher dimensions from 2 to 10-D. We show the radially averaged spectrum of the target (dotted blue) and our result (orange). Note, that plots appear increasingly discontinuous, as in higher dimensions the number of integer-frequency bins covering the normalized range* $(0, 4)$ *decreases exponentially.*

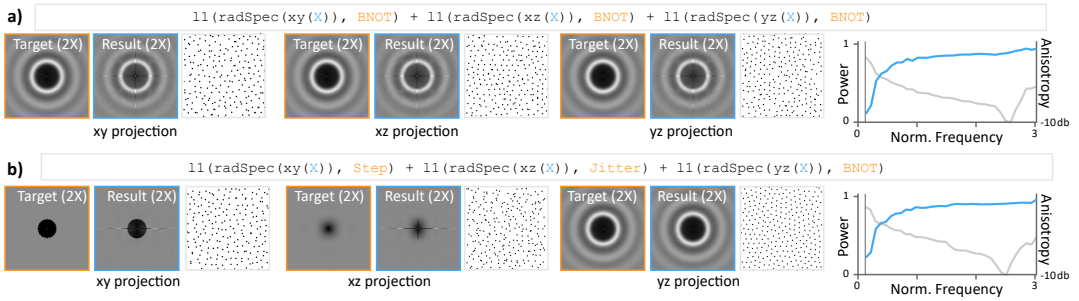


FIGURE 3.8: *Results for anisotropic point patterns with different properties in different subspaces. In each row, from left to right, we show the agenda, the spectra projected in* xy, xz *and* yz, *the radial mean (blue) as well as the variance (grey). In the first row, all targets differ in the second, they are the same.*

et al., 2016] in Fig. 3.6, a-d) as the radially averaged spectral profiles (orange and dotted blue) match the reference closely, in particular in the low-frequency regions. We see that also the 2D spectrum matches the reference while our learning was only supervised to produce a radial average. This shows that no additional anisotropy was introduced, further supported by the radial variance plots (grey).

Fig. 3.6, e–h, repeats the above experiment in the differential domain, where the target is the 1D point correlation function. Again both 1D and 2D PCF match. Note, that the PCF of Step BN [Heck et al., 2013] does only have a step in the 1D and 2D spectra (Fig. 3.6, c), not in the PCFs (Fig. 3.6, g). We learned reproducing this.

High-dimensional Isotropic

We explore the previous analysis for $n > 2$ in Fig. 3.7, for one specific important pattern, BNOT [De Goes et al., 2012]. Target spectra for higher dimensions were produced by scaling [Heck et al., 2013; Pilleboue et al., 2015] the two dimensional BNOT power spectrum computed over 1M samples. Thanks to our MC-based formulation that avoids reconstructing the high-D spectrum, we are able to measure such high-D spectra and even optimize for them.

We note that the results match the target, but this gets increasingly difficult in high-D. While at 5D, the BN peak is still distinct with a zero region and a peak at 1.0, at 10D the spectrum looks like jittered, but with a larger dark region. The only other concurrent work to achieve similar results is Spoke Darts BN [Mitchell et al., 2018], but at much higher algorithmic complexity (no code is changed to work in high dimensions for us) and compute cost (we compute this pattern in 150 ms). In particular, our method allows BN in subspaces as well, as explored next.

Subspace Results

Adapting correlations across projections with [Singh and Jarosz, 2017] and without [Keller, 2006; Dammertz and Keller, 2008] integrand knowledge have shown to provide better integration and anti-aliasing quality for relevant rendering problems. Projections have further uses far from integration, e. g., the arrangement of objects such that they appear well-distributed in both 3D space and in multiple 2D projections [Reinert et al., 2016] for visualization of 3D printing.

We now analyze agendas that ask for different spectra in different subspaces [Chiu et al., 1994; Reinert et al., 2016; Ahmed et al., 2016]. Our exemplary analysis is in $n = 3$ dimensions where three canonical 2-D subspaces, xy, xz and yz, exist. The agenda now sums three $\mathcal{L}_1$ losses in three subspaces in respect to three references. Results are shown in Fig. 3.8.

We see that our approach manages to produce a pattern that has the desired spectra in all projections Fig. 3.8, a. Note, that this would not be the case for a 3D BNOT pattern that is unaware of subspaces.

Taking it a step further (Fig. 3.8, b), we ask for different targets in different subspaces. Overall, each subspace achieves the desired target spectra, but with slight mutual concessions to be made: We see, that the xy projection that seeks to produce a Step BN spectrum shares the y projection with BNOT which is reflected in the long horizontal anisotropic line in our xy / Step BN spectrum. Along the vertical axis of xy, the anisotropy is due to the shared X projection with Jitter. Jitter (x, z) shares the z projection with BNOT. This manifests as the long horizontal anisotropic line in our Jitter (x, z) spectrum. Along the vertical axis of Jitter, the anisotropy is due to the shared X projection with Step BN.

We also show the radial average and the radial variance for both patterns (Fig. 3.8, b & c). The radial average looks like a modified jittered pattern for both: a small ramp and a constant range. As expected, for an anisotropic pattern, the variance is high.

Previous work has so far, by-construction, been only able to address special cases such as jitter in multiple spaces [Chiu et al., 1994], BN along canonical projections [Reinert et al., 2016] or combinations with LD patterns in 2D [Ahmed et al., 2016].

3.5.2 Monte Carlo Integration Convergence Analysis

In this section we look at an important application of point patterns: their use as Monte Carlo sampling. We will first look into the discrepancy after which we analyze integration convergence.

Discrepancy Our approach does not incentivize discrepancy as defined using classic metrics. Nonetheless, our patterns can have competitive discrepancy, as shown in Fig. 3.9, a, where we compute the box discrepancy (100 k samples) of several common patterns in 2D and 3D, as well as for some of ours. Common methods are random, jittered, and Sobol. Ours are BNOT and Step BN as used in Fig. 3.6.

For 2D, we see, the expected relations between common methods are present. We also see, that the discrepancy of BNOT and Step BN (dotted lines in Fig. 3.9 a, 2D) is performing better than jitter, with a discrepancy closer to Sobol, which performs best. Discrepancy is lower as a concession to spectral properties, which dominate MC convergence, as demonstrated next.

Variance We conclude our analysis by demonstrating our samples in a Monte Carlo rendering setting (Fig. 3.9, b). To this end, different scenes, e. g., integrands in 2D

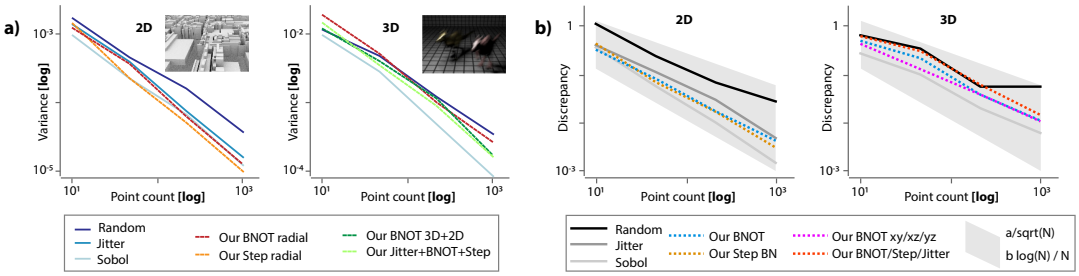


FIGURE 3.9: a) *Discrepancy analysis: The left sub-plot analyzes 2D, the right one 3D patterns. In each, the horizontal axis is log of point count and the vertical axis is the star discrepancy. The wedge bounds the theoretic limits.* b) *Variance analysis: The left sub-plot analyzes a 2D, the right one a 3D integration problem. In each plot, the horizontal axis is the log of the number of points and the vertical axis is variance for a single pixel inside the image. Different colors encode different methods.*

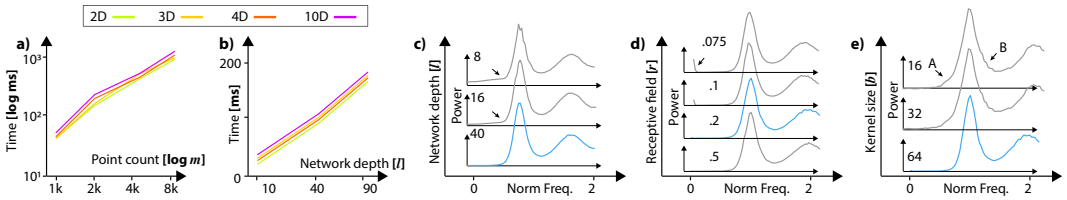


FIGURE 3.10: *Scalability analysis across numbers of points per time and error, network depth for time and error, receptive field and kernel size. Please see Sec. 3.5.3.*

(ambient occlusion) and 3D (pixel space and motion blur). We study our BNOT and Step BN variants (dotted lines). The sample variance of a single pixel at (100,100) was estimated over 200 realizations. A low variance is better, i. e., would render less noisy images.

In 2D, we find, that our learned methods (dotted in Fig. 3.9, b, 2D) performs better than random, jittered and Sobol, a typical industry standard pattern (solid, Fig. 3.9, b, 2D). For low sample counts, our method performs on-par and in particular our Step BN turns out to be most successful. Furthermore, ours scales to dimensions not possible to the target spectra-producing methods, such as 6D.

For 3D, Sobol performs best, but we see that extending BNOT from radial average (red dotted line in Fig. 3.9, b, 3D) to also include subspaces (green and blue dotted line in Fig. 3.9, b, 3D), results in better convergence. A combination including Step BN (light green dotted, Fig. 3.9, b, 3D) performs slightly better.

3.5.3 Scalability

Fig. 3.10 analyzes the scalability of our approach. Fig. 3.10, a shows compute time (y-axis, less is better) as a function of points (x-axis) and number of dimensions (different plots). We tested for a radially averaged BNOT target with $l = 40$ layers on a Tesla K40m GPU. We see that our approach can produce large patterns, up to 8.192 points. A point set with $1,024$ points requires ca. 100 ms to produce. In one second, we can produce sets in 2D to 10D of up to 8,192 points, indicating performance scales favorably with dimensionality.

Similarly, Fig. 3.10, b repeats the experiment, but for networks of varying depth. We see that the method scales linearly in all dimensions.

A key parameter of our network is its depth, i. e., the number of filter iterations. We study again the speed and error across different network depths in Fig. 3.10, c.

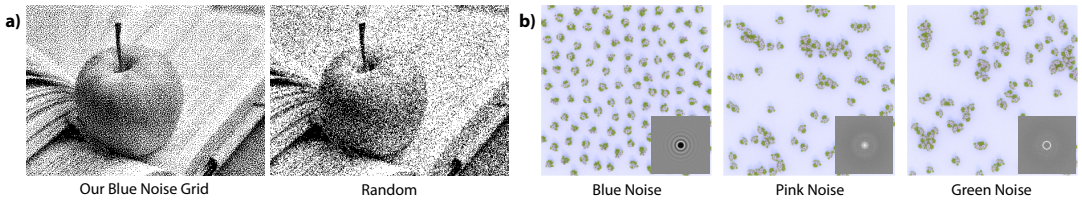


FIGURE 3.11: *Applications of our method.* a) *Dithering with our and a random mask.* b) *Object placement for three learned colors of noise.*

We see that time is roughly linear in the depth while quality saturates around the 45 levels we suggest.

An important parameter of our method is the size of the receptive field, which we vary in Fig. 3.10, d to see that the receptive field of $r = .2$ is a good compromise: Error for larger field saturates and smaller receptive fields produce low-frequency error (arrow in Fig. 3.10, d).

Finally, we look into the effect of filter kernels size (b) on the spectrum in Fig. 3.10,e, to find that quality saturate around the $b = 64$ we use. Smaller kernels smooth the profile (A) and create aliasing (B). Please note, that speed is not affected by kernel size (not shown), as it is a $\mathcal{O}(1)$ look-up.

We conclude that our method scales across different domains and is adjustable to trade quality and speed.

3.5.4 Applications

Our trained filters can efficiently be applied to problems such as dithering or object placement.

Dithering The ability to compute gridding masks (dithering patterns for rendering [Georgiev and Fajardo, 2016]) is demonstrated in Fig. 3.11, a. To achieve this, our framework does not require additional coding besides adding an enclosing `grid` operator that extracts the x dimensions from a 3D pattern keeping, y and z fixed. Explicit constructions of such masks can take considerable implementation effort (simulated annealing). This also demonstrates our framework's ability to handle different target spectra along different projections (in this case, blue noise along 1D and uniform for the rest).

Object placement We further demonstrate the capability of our framework to handle different target spectra. In Fig. 3.11, b, we trained for different colors of noise to procedurally place a flower object.

3.6 Conclusion

In this chapter we have proposed a framework to optimize for methods that turn random points without properties into point patterns with properties relevant for computer graphics tasks. Other than previous work that requires mathematical derivation and implementation effort, we simply state the forward model as a loss and rely on modern back-propagation to come up with a point pattern generation method. We have shown that several previous patterns can be emulated using our approach and in some cases even surpassed in terms of quality and/or execution speed.

Chapter 4

Minimal Warping: Planning Incremental Novel-view Synthesis

4.1 Introduction

Rendering images is time-consuming, in particular when complex visual phenomena such as motion blur or depth of field are needed, for spectral rendering, soft shadow or caustics, chromatic aberrations or when many images need to be produced, such as for light fields or high refresh rates. We observe, that images which such effects or those modalities are the sum of many pinhole images covering a "distribution domain" and that the information across this domain at the same time is highly redundant, presenting an opportunity for our method to exploit it.

One method to exploit this coherence is image warping: after an initial image is shaded, it is warped into several novel ones across the distribution domain and finally averaged. Warping is faster than re-rendering, mostly because geometric transformations and shading costs become decoupled from sampling the distribution effect visibility and is even shared over time or the angular domain. Still, warping itself has two main limitations: First, it can have limited speed in practice, especially when many and very different novel views are required. Second, it can only represent object or camera motion, and ignores other effects such as moving shadows or caustics or changes of wavelength. Our approach addresses these two shortcomings.

Addressing the first issue, our main observation is that small warps are easier than large warps. In particular, warps by a single pixel can be achieved by an effort not bigger than a conditional move from a tiny pixel neighborhood. Warps that do not change any pixels in a spatial or sampling-domain region are even better: they do not cost any time at all. To this end, instead of performing a large warp from the input image into each new view, we propose a system to plan the traversal of the warping space such that the warps become small. The resulting warping only needs to be done incrementally, i. e., by moving pixels by an amount that is spatially bounded or not at all.

Addressing the second issue, we generalize camera and object motion into a general notion of distribution flow that captures arbitrary reactions of the pixel position inside an image in response to changing any distribution coordinate, including area light sampling position or motion or animated caustics. To this end, we make two contributions: First we introduce shadow and spectral flow, that describe motions of pixels in response to motions of lights, occluders or receivers as well as changes in the spectral band. Second, to represent any arbitrary combination of flows, including shadow and spectral flow, we sample the flow at a low number of representative locations and extrapolate it to all other sample locations using radial basis functions.

The simplicity of our approach allows producing content that fuses arbitrary combinations of distribution effects (lens, time, area lights, spectrum, including

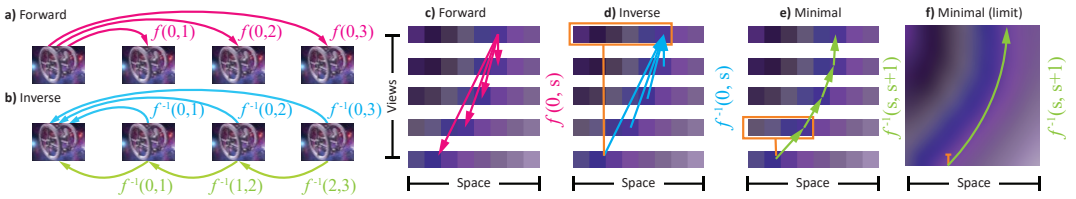


FIGURE 4.1: *Forward and inverse warping.* (a) *Starting from the original image at sample* $\mathbf{s} = 0$ *the warp* f *maps to new samples, here* 1, 2 *and* 3 (pink arrow). (b) *Commonly, inverse image warping seeks to create the inverse map* f^{-1} (blue arrow) *from the original image to all other images by inverting the forward warp. We show how the minimal warp* (green arrow) *between a known and a new inverse map is easier and faster to compute. The right part shows a comparison of different strategies for inverting many flows on a 1D image in a 1D sample space. The original image is shown in the top row. The forward flow is show in* (c). *Different samples are on the vertical axis, space is on the horizontal axis. Inverting the flow* (d) *using common approaches requires identifying the correct solution in a set of candidates* (orange box). *This can be accelerated using an iterative search [Yang et al., 2011; Bowles et al., 2012], but in practice compute time grows while quality is decreased when the warp distance grows. Our approach* (e) *only needs to search a small neighborhood to find the candidate. In the limit* (f) *this reduces to a single pixel.*

shadow or caustic motion) into arbitrary combinations of outputs (plain images, stereo pairs, light fields, higher temporal resolution). In particular, scenes with complex geometry and high shading cost benefit from this decoupling, producing images virtually free of sampling noise. Yet, we show how our approach, when using the sampling bounds we devise, does not introduce more error in comparison to a common warping-based solution while being much faster. At negligible planning cost, our approach is roughly three to four times as fast as drawing points or a forward-warping grid. For a one-megapixel resolution, minimal warping typically requires less than a millisecond, which is only four times slower than the theoretical optimum for any warping algorithm: Reading the forward flow from a texture and directly using it as the inverse flow for a lookup in another texture.

4.2 Overview

We will here give an overview of our approach, starting from the input and output (Sec. 4.2.1), motivating the need for minimal warping (Sec. 4.2.2) and giving the big picture of our pipeline (Sec. 4.2.3).

4.2.1 Input and Output

Input to our approach is a *root image* and a continuous distribution sample domain, e. g., lens, time and area light. Output is a small discrete set (e. g., a stereo pair) of images that are each convex combinations of discrete images in the sample domain.

The *root image* can be any RGB-D image, be it synthetic or acquired, taken from a representative sample, typically in the center of the domain. Highest quality results are obtained by combining our approach with layered depth image (LDI) rendering [Shade et al., 1998].

4.2.2 Minimal Warping

A *forward warp* from a source image to another view defines where every pixel is written to (Fig. 4.1, a). Such a forward warp is easily applied by drawing points [Zwicker et al., 2001; Sibbing et al., 2013], triangles [Mark et al., 1997] or a quad tree

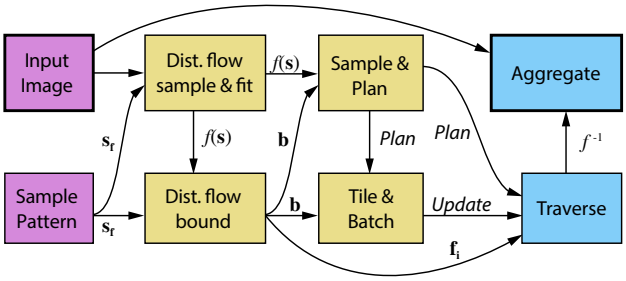


FIGURE 4.2: *Overview of our pipeline (see Sec. 4.2.3).*

[Didyk et al., 2010a]. Forward warping can produce holes or requires sophisticated filtering and sampling and does not parallelize well.

Our approach instead finds the *inverse warp* [Yang et al., 2011; Bowles et al., 2012], i. e., where every pixel is to be read from (Fig. 4.1, b). Why is finding an inverse flow hard? It is easy, if the mapping is constant over the image: When all pixels move five pixels to the right in the forward flow, every pixel just needs to look five pixels to the left in an inverse flow. If pixels move differently – and they do if the distribution space coordinates change in an arbitrary scene – the inversion is very hard. When found, however, it is used for sampling the texture and averaging the result per pixel [Haeberli and Akeley, 1990]. This avoids holes and fits modern GPUs well.

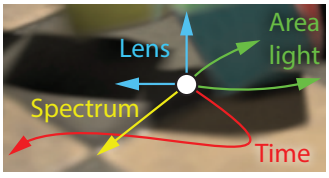
The differences of forward and inverse mappings are best visualized in 2D such as on the right of Fig. 4.1. The input image (top row) is mapped forward (pink arrows in Fig. 4.1, c) or backward (blue arrows in Fig. 4.1, d) to novel views (other rows). Instead of previous work that uses iterative search to find the inverse map between the input image and a new view, we proceed incrementally, allowing to work with minimal warps (Fig. 4.1, e). In the limit, the search radius becomes a single pixel (Fig. 4.1, f). For this to work, the approach requires planning.

4.2.3 Pipeline

Our approach has several steps (blocks in Fig. 4.2) and two main parts (colors in Fig. 4.2). The first part (yellow in Fig. 4.2) estimates distribution flow (Sec. 4.3) and plans the warping (Sec. 4.4.1 and Sec. 4.4.2), the second one executes this plan (Sec. 4.4.3, blue in Fig. 4.2). The first part relies on intelligence, involving function fitting, graph operations, optimizations, etc. and is mostly CPU-based. The second part is very simple and executed on the GPU by no more than massive conditional memcopies.

Distribution flow model Our approach proceeds in respect to a certain distribution flow model, which defines how a 3D scene position changes as a function of quantities such as, time, lens position, area light coordinate, wavelength, or index in the stereo image pair of light field image array. Before planning the minimal warps, we therefore first need to know how 3D distribution flow responds to changes in the corresponding distribution domain.

A simplified version of distribution flow is shown on the right: the arrows show motion of the central pixel in response to changes in the distribution sample domain. The image is a simplification, as we are interested in the *combined* effect of all dimensions, while the image only shows flow with respect to changes in a *single* variable.



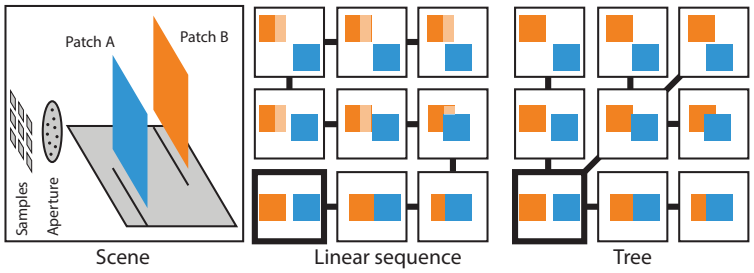


FIGURE 4.3: *"Wiping" effects in a linear sequence and a tree.*

We opt to sample the domain using a low-discrepancy
pattern of pilot samples and fit a radial basis function (RBF) model to the pixel motion. Output of this step is a model of how each pixel moves in 3D when any of its distribution parameter changes.

Planning The planning phase uses the distribution flow model to sample the distribution domain into images where pixels move at most by a single-pixel distance when moving from one image to its neighbor. The images are arranged into a *sample tree*, in which an edge is an inverse warp between two images. The tree is organized such that its traversal will always produce *minimal warps*. We define a warp from image A to image B to be *minimal*, if for every pixel in the output image B, it has not moved more than one pixel from its position in A.

An example is Fig. 4.3, rendering depth-of-field of two patches at different depth, sampled at 9 views for depth-of-field. Consider comparing two alternative sample graphs in the second and third column. The most simple sample graph could be to find the shortest path to connect all samples (Fig. 4.3, second column). Here, the long overall path can result in disocclusion (transparent in patch B that is behind patch A) propagating over the entire solution (referred to as "Wiping"). Instead, we arrange samples in a tree (Fig. 4.3, third column). Now, the disocclusions can only affect a single branch, or as in this case, no pixels at all. Intuitively, combining a tree-shaped flow that always expands and never shrinks with small warp lengths creates an efficient solution without occlusion problems. "Wiping" is the only reason, why our solution is not identical to the ground-truth inverse warp, i. e., an exhaustive search for the best matching pixel in the source image.

The ideal traversal would minimize disocclusions directly instead of only minimizing the path length. However, no obvious method exists to predict disocclusions in a complex forward flow in an LDI without executing it.

4.3 Distribution Flow

We call flow which arises from general distributions and their mutual combinations *distribution flow*, as it is a key component for our solution of the distribution rendering problem [Cook et al., 1987]. In this section we give details on this notion. First, we formally define the domain and the mapping (Sec. 4.3.1). Then, we explain how individual flow components are computed (Sec. 4.3.2). Finally, we show how complex distribution flow is efficiently and compactly represented by using radial basis functions (Sec. 4.3.3).

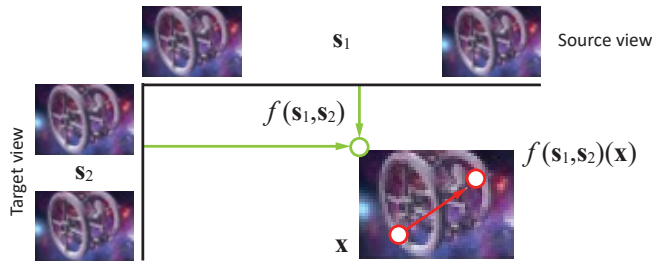


FIGURE 4.4: *Common forward flow* (red) *is a mapping from each image location* $\mathbf{x}$ *to a new location* $f(\mathbf{x})$. *We operate on a family of flows* $f(\mathbf{s}_1, \mathbf{s}_2)(\mathbf{x})$ *that depend on the source sample* $\mathbf{s}_1$ *and the target sample* $\mathbf{s}_2$ (green). *In this illustration, the sample space, which is high-dimensional, is depicted two-dimensionally for simplicity.*

4.3.1 Domain and Mapping

The *sample domain* $\mathbb{S}$ is a subset of the n_d-dimensional hypercube. Typically n_d is 2 or 3, but can also be higher. A *sample* is a position in the sample space and corresponds to an image (Fig. 4.4). The 3D motion in camera space is defined as a composed mapping $\mathbf{y} = f(\mathbf{s}_1, \mathbf{s}_2)(\mathbf{x}) \in \mathbb{S} \times \mathbb{S} \to (\mathbb{R}^2 \to \mathbb{R}^3)$ from the source and target sample $\mathbf{s}_1$ and $\mathbf{s}_2$ to a mapping of positions $\mathbf{x}$, visible in the original image, to a new 3D camera space position $\mathbf{y}$. The original image sample is denoted as $\mathbf{s} = 0$.

While the above notation describes the flow between two arbitrary samples $\mathbf{s}_1$ and $\mathbf{s}_2$ in distribution space, for the remainder of this section it is best thought of as the warping from one source image at 0 to another image with coordinate $\mathbf{s}$.

4.3.2 Flow Components

Taking a novel-view sample $\mathbf{s}$ induces a forward flow $f(0, \mathbf{s})$. In the following sections, we describe how to compute specific flow components individually. While object motion flow , camera flow , and aberration flow correspond to traditional published effects and are therefore only listed to make this chapter more self-contained, we give more details on two novel, very specific types of flow: The first one is *shadow flow* that predicts, how an image of a shadow changes when light, occluder or receiver move. The second is *caustic flow* that is applicable to photon-mapped caustics.

The process of flow estimation is best imagined as capturing `gl_Position` of a vertex shader that takes the distribution coordinate $\mathbf{s}$ as a parameter. Therefore, no re-rendering is necessary at any point during flow estimation. Furthermore, occlusions are not considered at this stage, since the warping step will take care of visibility configurations. The flow field is stored in a texture at full input image resolution.

In order to combine different flow components, the individual mappings are simply concatenated in light transport order, i. e., in the order they are listed below.

Object Motion Flow

Moving and deforming geometry causes a scene flow field [Vedula et al., 1999] that can be used to produce motion blur. In order to obtain the 3D position $\mathbf{x}(t)$ of the surface under a pixel, the rigid or non-rigid mesh deformation is sampled across time.

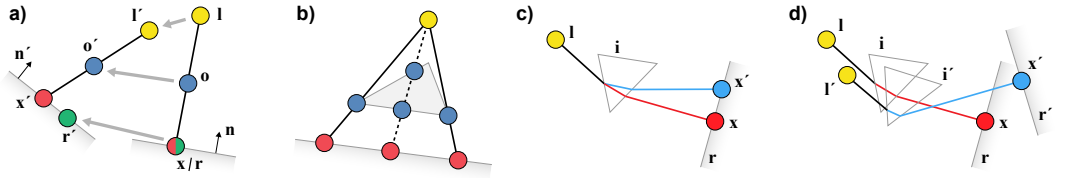


FIGURE 4.5: *The geometry of shadow* (a,b) *and caustic* (c,d) *flow.* (a) *A moving light source* $\mathbf{l}$*, occluder* $\mathbf{o}$*, and receiver* $\mathbf{r}$ *lead to a moving shadow position* $\mathbf{x}$*.* (b) *Only shadow silhouettes have a unique occluder.* (c) *A change of wavelength leads to caustic motion.* (d) *This can be combined with a moving light source* $\mathbf{l}$*, dispersive interface* $\mathbf{i}$*, and receiver* $\mathbf{r}$*. Dashed symbols denote the new configuration.*

Shadow Flow

Shadow flow predicts the motion of a 3D shadow point $\mathbf{x}$ for a point light source at position $\mathbf{l}$, blocked by an occluder at position $\mathbf{o}$ cast into a receiver position $\mathbf{r}$. To this end, we make the assumption, that the occluder does not change (the visibility graph does not change topologically, but deforms) and that the receiver is locally planar with normal $\mathbf{n}$. Under this condition, the new position $\mathbf{x}'$ can be computed by ray-casting an "updated" ray from the new light position $\mathbf{l}'$ through the new occluder position $\mathbf{o}'$ to the receiver plane with the new position $\mathbf{r}'$ and new normal $\mathbf{n}'$ (Fig. 4.5, a). This procedure is simply an analytical ray-plane intersection:

$$\mathbf{x}' = \mathbf{l}' + \mathbf{d}\frac{\langle \mathbf{l}' - \mathbf{r}', \mathbf{n}' \rangle}{\langle \mathbf{d}, \mathbf{n}' \rangle} \quad \text{with} \quad \mathbf{d} = \frac{\mathbf{o}' - \mathbf{l}'}{\|\mathbf{o}' - \mathbf{l}'\|_2}.$$

Since non-silhouette pixels do not have a unique occluder, the mapping from old to new shadow position is well defined only at shadow boundaries (Fig. 4.5, b). Therefore, we first compute shadow flow at the shadow silhouettes and then densify the resulting sparse flow field using pull-push.

Note, that rendering twice is no solution to the shadow flow problem, as it would explain how both shadows look like, but not which point has moved where. Shadow flow is applicable to temporal upsampling of moving lights, occluders or receivers as well as to sampling of area lights or the combination of both.

Caustic Flow

Caustic flow predicts where a single caustic $\mathbf{x}$ moves, if the wavelength or the position of the light $\mathbf{l}$, the dispersive interface $\mathbf{i}$, or the receiver $\mathbf{r}$ is changed (Fig. 4.5, c,d). Different from shadows at a single point that are caused by a single occluder, a caustic at a world position is caused by multiple reflectors. We will therefore have to resort to discrete derivatives and assume we use photon mapping to compute caustics: we simply send the same photon for a different light, time and spectral component and reproject it. Note, that in our approach this has to be done for a very low number of distribution pilot samples only, as described in Sec. 4.3.3. To reconstruct per-pixel flow, we use density estimation of distribution flows i. e., every photon does not splat its color, but the change of position, relative to the distribution coordinate, e. g., time.

Since this produces noisy estimates, we robustify it by, first, setting flow with a magnitude larger than a threshold to zero, and second, by performing median filtering. The flow is finally densified using pull-push.

Camera Flow

Camera flow produces motion blur induced by camera motion as well as depth of field from a thin lens. The 3D object position, which may be altered by the previously described flows, is projected using a lens-time-sampled camera matrix $A(u, v, t)$ [Haeberli and Akeley, 1990].

Aberration Flow

For transverse chromatic aberrations (Fig. 4.11) a typical forward flow is given by the second-order radial lens distortion model

$$f(0, \mathbf{s})(\mathbf{x}) = \begin{pmatrix} \mathbf{x}_x \left(1 + \lambda_\mathbf{s} \left(\alpha r + \beta r^2\right)\right) \\ \mathbf{x}_y \left(1 + \lambda_\mathbf{s} \left(\alpha r + \beta r^2\right)\right) \\ \text{depth}(\mathbf{x}) \end{pmatrix},$$

where $\lambda_\mathbf{s}$ denotes the wavelength of sample $\mathbf{s}$, $r = \sqrt{\mathbf{x}_x^2 + \mathbf{x}_y^2}$ denotes the distance from the image space position $\mathbf{x}$ to the image center $\mathbf{x}_0 = \mathbf{0}$, and α, β are free parameters to control the amount and shape of the radial distortion.

4.3.3 Representing Distribution Flow

The distribution flow is potentially high-dimensional and partly expensive to evaluate, making it impractical to sample for any cubature-type approach. In contrast, we will perform a cubature-like sampling to produce individual view samples in Sec. 4.4.1. This is feasible, since the amount of work needed for creating a view sample via minimal warping is in the order of magnitude of a texture copy operation.

To find the distribution flow for every pixel, we assume it to be smooth across the sample domain $\mathbb{S}$. Note, that this does not assume spatial smoothness of either the image itself or the flow field or the inverse flow field, which need to tackle occlusions.

Inspired by quasi-Monte Carlo rendering we create a low number K of pilot samples $\mathbf{s}_f$ with low discrepancy (Halton pattern with leaping [Robinson and Atcitty, 1999]) and sample the flow at these pilot locations, avoiding the curse of dimensionality and still covering the domain well. We include an additional deterministic sample at $\mathbf{s} = 0$ to ensure that the original image will be interpolated exactly. We did not see evidence that more than $K = 12$ samples provide a change in the scenes we explored. Next, we fit a radial basis function (RBF) model

$$f(0, \mathbf{s})(\mathbf{x}) = \sum_i^K \mathbf{w}_i(\mathbf{x}) \phi(\|\mathbf{s} - \mathbf{s}_{f,i}\|_2) \tag{4.1}$$

with $\mathbf{s}_{f,i}$ being the i'th pilot sample. We use the inverse multiquadric radial basis function $\phi(r) = \left(r^2 + a^2\right)^{-\frac{1}{2}}$ with scale factor $a = 2n_d$. The weights $\mathbf{w}_i(\mathbf{x}) \in \mathbb{R}^3$ are determined by solving the linear system

$$\sum_i^K \mathbf{w}_i(\mathbf{x}) \phi(\|\mathbf{s}_{f,i} - \mathbf{s}_{f,j}\|_2) = f(0, \mathbf{s}_{f,j})(\mathbf{x}), \quad \text{for } 1 \le j \le K.$$

This is done by first performing a QR decomposition of the $K \times K$ system matrix (which is the same for all pixels) and with this then cheaply and robustly solving the system for each pixel in parallel.

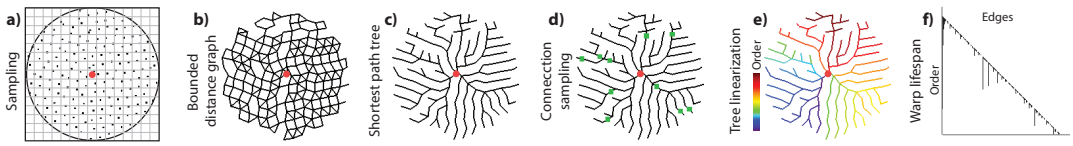


FIGURE 4.6: *Overview of six steps for sample planning, here for a two-dimensional lens sample space example.*

After this step, the flow from the root to any sample can be approximated by K RBF evaluations and a dot product.

4.4 Minimal Warping

The central idea of this work, is to efficiently find the inverse flow $f^{-1}(0, \mathbf{s}_3)$ from the original image to many $\mathbf{s}_3$ by recursively re-using solutions $f^{-1}(\mathbf{s}_1, \mathbf{s}_2)$ computed previously. The inverse flow is not the inversion of the entire mapping. Instead, it can only explain where a 2D position in the novel images was in the original image.

We will now explain the four main steps of our algorithm in detail. Planning (Sec. 4.4.1) produces a set of samples and a traversal order that allows for minimal warps. Tiling and Batching (Sec. 4.4.2) determines which image regions for which samples can be skipped without introducing errors. Warping (Sec. 4.4.3) performs the flow inversion, and Aggregation (Sec. 4.4.4) produces the final output.

4.4.1 Sample Planning

Input to the planning is the sample domain $\mathbb{S}$ and an RBF distribution flow. Output is a discrete set of samples S and a view traversal order. Planning proceeds in seven steps (Fig. 4.6): Bounding distribution flow *(i)* and sampling *(ii)* produce a set of samples with a carefully determined density that allows for minimal warps. A bounded distance graph *(iii)* in conjunction with a shortest-path tree algorithm *(iv)* and connection sampling *(v)* augment the samples with a connectivity structure. Tree linearization *(vi)* produces the final traversal order. Optionally, a traversal stack *(vii)* can be created to significantly reduce the memory requirements of the Warping step.

Bounding distribution flow In order for minimal warping to work we need to know how a change of a sample domain coordinate translates into pixel motion in image space. More specifically, we want to know how many samples are needed per sampling dimension so that f moves at most p pixels in the image when moving from one sample to the next. We denote this quantity $\mathbf{b} \in \mathbb{R}^{n_\mathrm{d}}$. The total number of samples is then $n_\mathrm{s} = \prod \mathbf{b}_i$. A usual value for p is 1 or 2 pixels.

For determining a conservative estimate of $\mathbf{b}$ we are interested in the maximum rate of change of f with respect to each sampling dimension. Therefore,

$$\mathbf{b}_i = \frac{1}{p} \max_{\mathbf{x}} \max_{\mathbf{s} \in \mathbb{S}} \left\| \left(\frac{\partial f^{(1)}(0, \mathbf{s})(\mathbf{x})}{\partial \mathbf{s}^{(i)}}, \frac{\partial f^{(2)}(0, \mathbf{s})(\mathbf{x})}{\partial \mathbf{s}^{(i)}} \right)^T \right\|_2, \tag{4.2}$$

where $\mathbf{s}^{(i)}$ denotes the i'th component of $\mathbf{s}$ and $f^{(1)}$ and $f^{(2)}$ are the horizontal and vertical component of the flow, respectively. The depth component of f is not necessary at this point, because it does not give any information about image space motion. The maximum over $\mathbf{s}$ is found using gradient ascent, starting at every pilot sample

$\mathbf{s}_f$, in parallel for all pixels. The maximum over all pixels $\mathbf{x}$ is found using a max MIP map.

Sampling Sampling (Fig. 4.6, a) considers the entire continuous domain $\mathbb{S}$ and produces a discrete sample set $S = \{\mathbf{s}_i\}$. To this end, the sample domain is discretized into a grid of size $\mathbf{1}/\mathbf{b}$ (where / denotes element-wise division) and one sample is produced per grid cell. Following the above, samples are placed exactly such that pixels move only p pixel distances when going from one sample to the next for the entire sampling domain. Additionally, every sample is subject to correlated multi-jittering [Kensler, 2013]. Our approach naturally lends to such a sampling pattern as it exhaustively covers the domain in all dimensions by design to not miss any feature.

Bounded distance graph Key to our planning is to arrange the samples into a graph (Fig. 4.6, b). In this graph, the samples $\mathbf{s}_i$ are vertices. Edges are created between two vertices if their sample space distance corresponds to less than p pixels in image space. The distance between samples i and j is measured according to a $\mathbf{b}$-weighted norm that converts the Euclidean sample distance into units of pixel motion, so $d_{ij} = ||\mathsf{B}(\mathbf{s}_i - \mathbf{s}_j)||_2$, where B is a diagonal matrix scaling dimension k by $\mathbf{b}_k$. The graph is computed in parallel for all $n_s \times n_s$ candidate edges and is very sparse with an average outdegree of 1.8.

Shortest-path tree Next, the graph is converted into a shortest-path tree, with the input sample $\mathbf{s} = 0$ as a root, using a breadth-first search in linear time (Fig. 4.6, c). For each node during search we select the child being explored next at random, which leads to a smaller traversal stack (explained below).

Connection Sampling Occasionally, due to the jittered sampling pattern, tree edges correspond to a pixel motion slightly larger than p. Therefore, we simply add a sample in the middle of each such edge (Fig. 4.6, d).

Tree linearization Tree linearization turns the tree into a sequence of edges (indices of parent and child) as encountered in a depth-first, pre-order traversal (Fig. 4.6, e). Depth-first warping is preferred over breadth-first warping that would also allow for minimal warping, but requires to store a much larger number of intermediate inverse warps when using a traversal stack as explained below. No particular order is necessary among children. However, to optimize progressiveness, optionally, when more than two children are present, we employ a farthest-first traversal of the children. As the representative position of a child we choose the mean position of all nodes in the child's sub-tree. The farthest-first traversal avoids exploring similar sub-trees sequentially and encourages the exploration of diverse solutions first. The end-outcome is not affected by this operation, only when executing parts of S, e. g., for progressive preview.

Traversal stack While the procedure described above is sufficient to execute a minimal warping plan, it still requires to have all previously visited nodes in memory as they can appear as the parent of any child. If we are interested in creating distribution effects, there is no need to store all warped images. Instead, we can perform a moving average to achieve the same effect. In this step, which is optional, the traversal is augmented with additional information, such that only a small stack of active inverse

warps needs to be remembered when executing the minimal warping plan. Therefore, the space complexity of our approach depends on the depth of the traversal tree and not on the total number of its vertices.

A warped sample needs to be remembered only, if it will be needed as a source sample for a minimal warp at a later point in the traversal (Fig. 4.6, f). Whenever a new sample $\mathbf{s}_i$ has been created from a source (i. e., parent) sample $\mathbf{s}_{i-1}$, two configurations may occur: *a)* If $\mathbf{s}_i$ is the last child of $\mathbf{s}_{i-1}$ in the traversal order, $\mathbf{s}_{i-1}$ can be forgotten. *b)* If $\mathbf{s}_i$ is not a leaf, $\mathbf{s}_i$ needs to be stored as a source sample for one or more later warps. Since the sample ordering originates from a pre-ordered, depth-first traversal of a tree, a simple stack data structure is sufficient to encode this behavior. After the last occurrence of a sample $\mathbf{s}_{i-1}$ as the source for a minimal warp, the sample can be popped from the stack. When a newly created sample $\mathbf{s}_i$ will be needed as a source for a later warp, it is pushed to the stack. Following this procedure, the source sample for the current minimal warp will always be the top of the stack while executing the traversal plan.

4.4.2 Tiling and Batching

Depending on the distribution domain and its associated flow, different image regions move at different speeds. Some regions are even completely unaffected by the sampling and do not move at all. We exploit this fact by subdividing the image into tiles (for execution coherence) and determining a distinct update pattern for each tile.

When using the max MIP map for solving Equation 4.2, we can simply read the MIP map at a lower level to get a vector $\mathbf{b}^{(i)}$ per tile. A typical tile size is 16×16 pixels. The values of $\mathbf{b}^{(i)}$ cannot be used as a representative sampling density measure for the tiles, since they are only estimating forward flow. A tile in an inverse warping framework can only be skipped, if nothing maps to it. In order to get an estimate of the inverse flow bounds $\widetilde{\mathbf{b}}^{(i)}$, we first compute a per-pixel axis-aligned bounding box (AABB) of the flow (Fig. 4.7, a). The four sides of the AABB are determined as

$$\begin{aligned} \text{left/right} &: \min_{\mathbf{s}\in S} / \max_{\mathbf{s}\in S} f^{(1)}(0,\mathbf{s})(\mathbf{x}), \\ \text{bottom/top} &: \min_{\mathbf{s}\in S} / \max_{\mathbf{s}\in S} f^{(2)}(0,\mathbf{s})(\mathbf{x}), \end{aligned}$$

using gradient descent and ascent, respectively, in parallel for all pixels. Then, we intersect each tile with the AABBs of all other tiles and compute, in parallel for all tiles, $\widetilde{\mathbf{b}}^{(i)} = \max_{j\in\Omega_i} \mathbf{b}^{(j)}$, where Ω_i is the set of all indices corresponding to tiles whose AABB intersects tile i. The maximum is performed component-wise. Tiles with $\widetilde{\mathbf{b}}^{(i)} = \mathbf{0}$ are not affected by the distribution sampling. Consequently, they do not need to be processed at all. As a side effect, this achieves scalability in regard to scenes with high depth complexity by avoiding spending too much work on almost-empty LDI layers.

Since we cannot afford to compute and store an update pattern for each tile, all remaining active tiles are clustered by their $\widetilde{\mathbf{b}}$ vectors (Fig. 4.7, b). We use k-means clustering using scattering and blending [Dong et al., 2009] ($k = 20$ in all our experiments). To produce the final update pattern per cluster (Fig. 4.7, c), we iterate over all samples $\mathbf{s}$ for all tiles in parallel: If in any sampling dimension the distance between the current sample and the sample at which the cluster had its last update is larger than the cluster's value $\mathbf{1}/\widetilde{\mathbf{b}}$, the cluster needs an update at the current sample's parent in the traversal tree. Once the update pattern is created, all

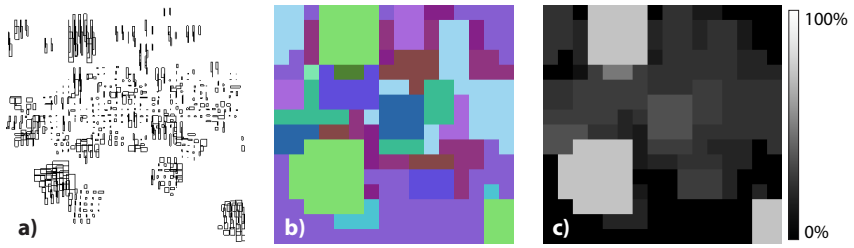


FIGURE 4.7: *Tiling and batching is done by computing axis-aligned bounding boxes of the flow* (a) *to perform clustering of tiles* (b)*. In* (c) *the update frequency for individual tiles during sample space traversal is shown. In this motion blur example most tiles need an update only at every third sample or less often.*

that needs to be done for each tile during the sample space traversal is to look up if it needs an update based on its cluster ID. During aggregation, this update pattern will be taken into account.

4.4.3 Warping

The warp itself is executed for every sample in two steps: Requesting evaluates the forward flow and computes necessary auxiliary information. Searching inverts this forward flow based on a previously warped sample.

Requesting

Requesting computes $f(0, \mathbf{s}_i)$, the camera-space 3D forward flow from the original image to sample $\mathbf{s}_i$ of every pixel seen in the original image. This is done by evaluating Equation 4.1.

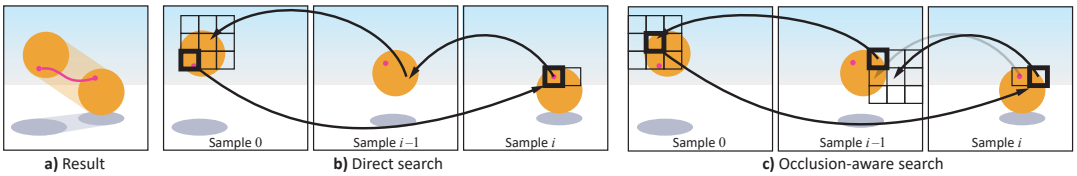
From the forward flow field we additionally compute a magnification field $m(0, \mathbf{s}_i)(\mathbf{x})$ at sample $\mathbf{s}_i$. This field contains the larger absolute eigenvalue of the Jacobian

$$J(0, \mathbf{s}_i)(\mathbf{x}) = \begin{pmatrix} \frac{\partial f^{(1)}}{\partial \mathbf{x}^{(1)}} & \frac{\partial f^{(1)}}{\partial \mathbf{x}^{(2)}} \\ \frac{\partial f^{(2)}}{\partial \mathbf{x}^{(1)}} & \frac{\partial f^{(2)}}{\partial \mathbf{x}^{(2)}} \end{pmatrix} (0, \mathbf{s}_i)(\mathbf{x}),$$

where, again, $\cdot^{(1)}$ and $\cdot^{(2)}$ denote horizontal and vertical components, respectively. The entries of the Jacobian are computed numerically from neighboring pixels using central differences. A magnification field value of 1 indicates that the forward flow merely induces translation or rotation locally in image space. In a minification or magnification condition, this value is smaller or larger than 1, respectively. The magnification field is later used to appropriately deal with those two conditions. Note that neither condition will increase the number of elements required to search for flow inversion, but will only adapt the thresholds for deciding which candidate is better than a different one.

Searching

Next, we explain how the plan can be executed by performing minimal warps. A minimal warp is essentially a small search. As we will explain, the search can be performed *directly* or in an *occlusion-aware* fashion. Finally we will show, how, while we work on an integer nearest neighbor field (NNF), an optional sub-pixel accurate search allows to read the input image at fractional coordinates.



a) Result **b)** Direct search **c)** Occlusion-aware search

FIGURE 4.8: *Details of our searching procedure. Consider a yellow ball with a pink dot casting a shadow moving from left to right and rotating to produce motion blur* (a). *Our method inverts the flow from sample* $i-1$ *to sample* i. *For the direct search (Alg. 1), in sample* i, *for one pixel* (thick box), *the inverse flow in sample* $i-1$ *is looked up. Using this result, a neighborhood* $[-p,\dots,p]^2$ *is searched in sample* 0, *i. e., the original image, to find the pixel that best maps to the current sample* (again, thick box). *This procedure works well, except at occlusions, which require an occlusion-aware search (Alg. 2) instead* (c). *Here the pixel of interest in sample* i (thick box) *is an occlusion, i. e., the same pixel in sample* $i-1$ *belongs to the background. The solution is to perform two searches: The first one finds the best position in sample* $i-1$, *the second one proceeds as in direct search. Note, that in the case of simple 2D motion like in this example, the second search is not necessary, because the first search finds the best position immediately. In case of perspective 3D motion, the second search prevents error accumulation.*

Direct search The direct search is illustrated in Fig. 4.8, b. Searching will produce the inverse warp of the current sample $f^{-1}(0,\mathbf{s}_i)$ from the inverse warp of the previous sample $f^{-1}(0,\mathbf{s}_{i-1})$. The planning step has assured that for every $\mathbf{x}$, the solution $f^{-1}(0,\mathbf{s}_i)(\mathbf{x})$ is in the set $\{f^{-1}(0,\mathbf{s}_{i-1})(\mathbf{x}\oplus[-p,\dots,p]^2)\}$, where $\oplus$ denotes Minkowski summation i. e., the set of all points in a two-dimensional integer lattice of side length $2p+1$ around that point. Intuitively, this says that the correct solution for the current sample is known to be a very nearby other pixel in a previous sample. Therefore, for every pixel in the output, all that is required is a two-dimensional loop over its $(2p+1)^2$ neighboring pixels.

Input : Inverse flow $f^{-1}(0,\mathbf{s}_{i-1})$ at previous sample,
Forward flow $f(0,\mathbf{s}_i)$ for new sample.
Output: Inverse flow $f^{-1}(0,\mathbf{s}_i)$ and depth at new sample.
forall *pixels* $\mathbf{x}$ *in* $f^{-1}(0,\mathbf{s}_i)$ **do**
  result($\mathbf{x}$) := vec3(0, 0, ∞);
  vec2 $\mathbf{u}$:= $f^{-1}(0,\mathbf{s}_{i-1})(\mathbf{x})$
  forall $\mathbf{y}\in[-p,\dots,p]^2$ **do**
    vec3 $\mathbf{t}$:= $f(0,\mathbf{s_i})(\mathbf{u}+\mathbf{y})$;
    if $\|\mathbf{x}_{xy}-\mathbf{t}_{xy}\|_\infty < m(0,\mathbf{s}_i)(\mathbf{x})$ *and* $\mathbf{t}_z \le \text{result}_z(\mathbf{x})$ **then**
      result($\mathbf{x}$) := vec3($\mathbf{u}+\mathbf{y}$, $\mathbf{t}_z$);
    end
  end
end
return result;

Algorithm 1: Direct search.

This can be implemented in a very simple and shader-friendly way as seen in Alg. 1. The outer loop is parallel over all pixels. The small inner loop is performed by a fragment shader. It merely requires visiting the neighboring pixels and performing two tests: If the forward flow maps to the current pixel, and if yes, if it is closer to the camera than any other pixel that previously also mapped there. The winner is the new inverse flow. The acceptance criterion for the decision whether a pixel maps to the current one depends on the minification or magnification m. In a magnification condition ($m>1$), the forward flow is allowed to map to a location further away from

the pixel center to be accepted, avoiding holes in the warped image. In a minification condition ($m < 1$), the tighter threshold leads to a more accurate selection of the best match.

Input : Inverse flow $f^{-1}(0, \mathbf{s}_{i-1})$ at previous sample,
Forward flow $f(0, \mathbf{s}_i)$ for new sample.
Output: Inverse flow $f^{-1}(0, \mathbf{s}_i)$ and depth at new sample.
forall *pixels* $\mathbf{x}$ *in* $f^{-1}(0, \mathbf{s}_i)$ **do**
  result($\mathbf{x}$) := vec3(0, 0, ∞);
  vec3 $\mathbf{u}$:= vec3(0, 0, ∞);
  forall $\mathbf{y} \in [-p, \ldots, p]^2$ **do**
    vec2 $\mathbf{r} := f^{-1}(0, \mathbf{s}_{i-1})(\mathbf{x} + \mathbf{y})$;
    vec3 $\mathbf{q} := f(0, \mathbf{s_i})(\mathbf{r})$;
    if $\|\mathbf{x}_{xy} - \mathbf{q}_{xy}\|_\infty < m(0, \mathbf{s}_i)(\mathbf{x})$ *and* $\mathbf{q}_z \leq \mathbf{p}_z$ **then**
      $\mathbf{u}$:= vec3($\mathbf{r}$, $\mathbf{q}_z$);
    end
  end
  forall $\mathbf{y} \in [-1, 0, 1]^2$ **do**
    vec3 $\mathbf{t} := f(0, \mathbf{s_i})(\mathbf{u}_{xy} + \mathbf{y})$;
    if $\|\mathbf{x}_{xy} - \mathbf{t}_{xy}\|_\infty < m(0, \mathbf{s}_i)(\mathbf{x})$ *and* $\mathbf{t}_z \leq \text{result}_z(\mathbf{x})$ **then**
      result($\mathbf{x}$) := vec3($\mathbf{u}_{xy} + \mathbf{y}$, $\mathbf{t}_z$);
    end
  end
end
return result;

Algorithm 2: Occlusion-aware search. The gray part is identical to the non-occlusion-aware variant.

Occlusion-aware search The above direct search procedure fails, if the target pixel was occluded in the source sample. In this situation, the inverse flow from the previous sample is an inaccurate initial guess of the current inverse flow. But since also occlusion boundaries can only have moved within $[-p, \ldots, p]^2$, the problem can be solved by an additional small search in $f^{-1}(0, \mathbf{s}_{i-1})(\mathbf{x})$ for the best initial estimate, as seen in Fig. 4.8, c and defined in Alg. 2. After this search, the algorithm proceeds as in the non-occlusion-aware version. All results in this chapter were produced using occlusion-aware search.

Sub-pixels accuracy The above procedure searches a low number of *discrete* choices. The continuous forward flow, however, maps every discrete pixel in the initial sample to a continuous sub-pixel location. Sub-pixel accurate inverse flow is found by fetching the four pixels around the discrete optimum and computing the continuous location $\mathbf{y}$ that minimizes the cost of pyramid matching [Bailey, 2003]. In practice, we never remember the continuous sub-pixel coordinate in floating point precision, but a discrete NNF with integer coordinates to avoid the accumulation of floating point errors.

Sorting

When using LDIs, the sparsity naturally occurring in the back layers can be used to prevent wiping even more. On a per-pixel basis, values are simply pushed back as

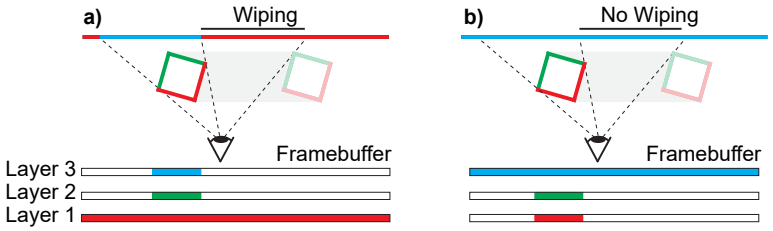


FIGURE 4.9: *Per-pixel sorting of layers prevents wiping, here illustrated for a box moving in front of a planar background.*

far as possible in the layer ordering, as illustrated in Fig. 4.9. This is done once per input LDI. All LDI results in this chapter were produced using sorting.

Implementation

Requesting and searching are implemented in parallel for all pixels in all tiles. Our prototype is implemented in the OpenGL shading language. The plan is stored in a vertex buffer object and executed by layered rendering, where the samples correspond to layers. A geometry shader emits quads for all tiles that require updating. Synchronization has to be performed, such that all tiles in one layer are finished before a new layer can start.

4.4.4 Aggregation

Once the inverse flow for a view sample has been found, it can be used to perform a simple texture lookup of the input image to create the associated novel view. After a sample has been produced, it can contribute to a single, or multiple output images. Let n_o denote the number of output images. In the simplest case, the output is a single image, and all samples are averaged. The other extreme is temporal up-sampling: here multiple samples are produced and each one is a result image. In general, multiple samples can contribute to multiple output images. In general, the aggregation of n_s samples into n_o images can be described as an *aggregation matrix* denoted as $A \in \mathbb{R}^{n_s \times n_o}$. Fig. 4.10 shows a visualization of A. The aggregation is performed point-wise for all output modalities. When not using LDIs as input to the system, disocclusions occur naturally. During aggregation, these undefined regions are simply ignored.

4.5 Results and Discussion

Here we present qualitative (Sec. 4.5.1) as well as quantitative (Sec. 4.5.2) results of our approach, and discuss design choices (Sec. 4.5.3) and limitations (Sec. 4.5.4).

4.5.1 Qualitative Results

Here we show several applications of planning minimal warping.

Temporal up-sampling Besides distribution effects, a typical application of warping is temporal up-sampling (frame rate conversion) [Didyk et al., 2010b; Bowles et al., 2012]. This is challenging in the presence of complex geometry and motion. In Fig. 4.10 we show a sequence of several moving characters, undergoing complex accelerating local motion between two key-frames (left- and rightmost image).

FIGURE 4.10: *Temporal up-sampling from two key-frames to in-between images with a* 360° *degree tent shutter. Insets show pairs of our interpolation* (top) *and the rendered reference* (bottom)*. The aggregation matrix* $A \in \mathbb{R}^{256\times3}$ *is shown right (large weights are darker).*

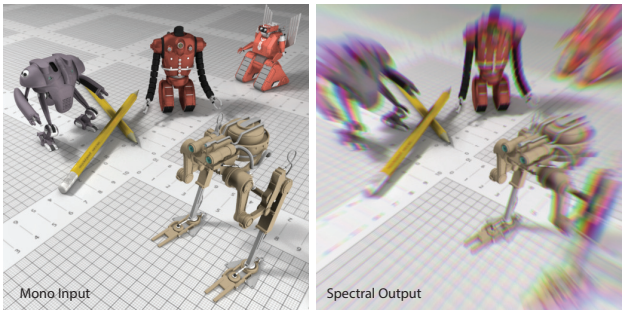


FIGURE 4.11: *Chromatic aberrations produced by our method.*

Stereo-to-light field A typical application requiring many warps is conversion of stereo content to light fields, in particular, when correct inter-view filtering is required to avoid aliasing [Zwicker et al., 2006]. In Fig. 4.12, depth is created by finding correspondences from the stereo pair, which is then used to create many virtual views that are carefully combined using a custom aggregation matrix to avoid inter-view aliasing.

Depth-enabled photo effects Finally, we demonstrate the quality and time of adding several effects to acquired RGB-D data [Silberman et al., 2012] in Fig. 4.13 using our approach.

4.5.2 Quantitative Results

Our approach is compared to different competitors in terms of computing different effects. As competitors, we consider *i)* point splatting, *ii)* grid warping with a cell size of one pixel [Mark et al., 1997], *iii)* quad tree-warping [Didyk et al., 2010a], and *iv)* a ray-traced reference. We did not consider iterative gathering methods in this comparison, because the complex flow fields with wide baselines lead to non-convex

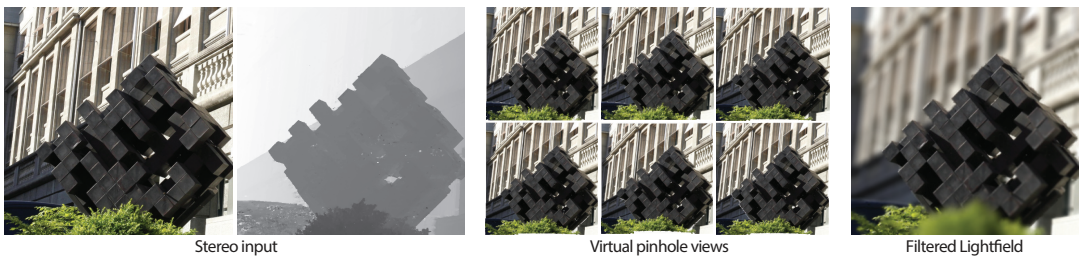


FIGURE 4.12: *Conversion of a stereo image* (left) *into a light field* (right) *– here showing a single new view. Our approach creates many virtual views* (center) *that are carefully combined to provide inter-view anti-aliasing [Zwicker et al., 2006].*

FIGURE 4.13: *Results of our method applied to acquired RGB-D data from the NYU dataset [Silberman et al., 2012].* a) *and* b)*: Depth of field, 256 samples.* c)*: Motion blur from camera motion, 64 samples.* d)*: Motion blur from camera motion, 128 samples.* e)*: Original images.*

energies to minimize, requiring these methods to be initialized, i. e., with a quad tree [Bowles et al., 2012]. This shows, that any converging search method can in principle not be faster than our approach which is already 2 to 3 times faster than quad tree-warping. All competitors had the same LDIs as input.

The effects tested are *i)* depth-of-field, *ii)* motion blur, *iii)* soft shadows, *iv)* spectral caustics, and *v)* combinations of the above. Fig. 4.14 shows images produced from our approach as well as insets made from our approach and the ray-tracing reference. The results were produced on a Nvidia Geforce GTX 980Ti with an Intel Xeon E5-1607 CPU at a resolution of 1024×1024 pixels and are summarized in Tbl. 4.1. The given timings include all steps described in this chapter, except for the "Prism" scene, where the spectral photon tracing (Sec. 4.3.2) was done using an external application. We performed same-quality comparisons, as can be seen from the almost identical SSIM values. We observe that our approach can produce same-quality images in significantly less time than common approaches using warping. Additionally, our novel shadow and spectral flow formulations allow our approach to produce a more versatile range of distribution effects.

In Tbl. 4.2 we give timings for the individual processing stages of our method. It can be seen that the planning stage usually constitutes only a small fraction of the total amount of work. It is furthermore noticeable that the timing for the warping is almost in the same order of magnitude as the aggregation step i. e., summing the novel views.

Tbl. 4.3 shows equal-time and equal-quality comparisons of our approach to a path tracing solution created with Renderman 20.10. It can be seen that path tracing, which has to perform shading evaluation for each ray, produces results of significantly less quality in the same time and needs considerably longer to compute equal-quality results. This is true even if the time to shade a pinhole image i. e., one sample per pixel, which is the input to our system (timings are given in the *Shading* column of Tbl. 4.2), is taken into account.

TABLE 4.1: *Efficiency and visual quality of different methods (columns) on different scenes (rows). Factor is time relative to our solution (smaller is better). Similarity is measured in SSIM [Wang et al., 2004] relative to a ray-tracing reference (larger is better). Distribution effects marked with an asterix cannot or can only partially be reproduced by the competitors.*

Scene	Effect	Point			Grid			Quad-tree			Ours	
		Time	Fac.	Sim.	Time	Fac.	Sim.	Time	Fac.	Sim.	Time	Sim.
Dancers	MB	4.1 s	×4.2	.93	2.5 s	×2.6	.93	2.1 s	×2.1	.92	**1.0 s**	.92
Garden	DOF	84.8 s	×2.1	.91	99.5 s	×2.4	.90	104.9 s	×2.6	.88	**41.2 s**	.90
Bouncing Cubes	MB + Shadow MB*	35.9 s	×4.8	.88	24.3 s	×3.2	.86	15.2 s	×2.0	.85	**7.5 s**	.86
Vehicle	Soft Shadows*	-	-	-	-	-	-	-	-	-	**40.0 s**	.96
Space Station	MB + DOF	1350.1 s	×3.5	.93	1132.5 s	×2.9	.92	1035.0 s	×2.7	.91	**387.7 s**	.93
Prism	Spectral Caustics*	-	-	-	-	-	-	-	-	-	**4.2 s**	-

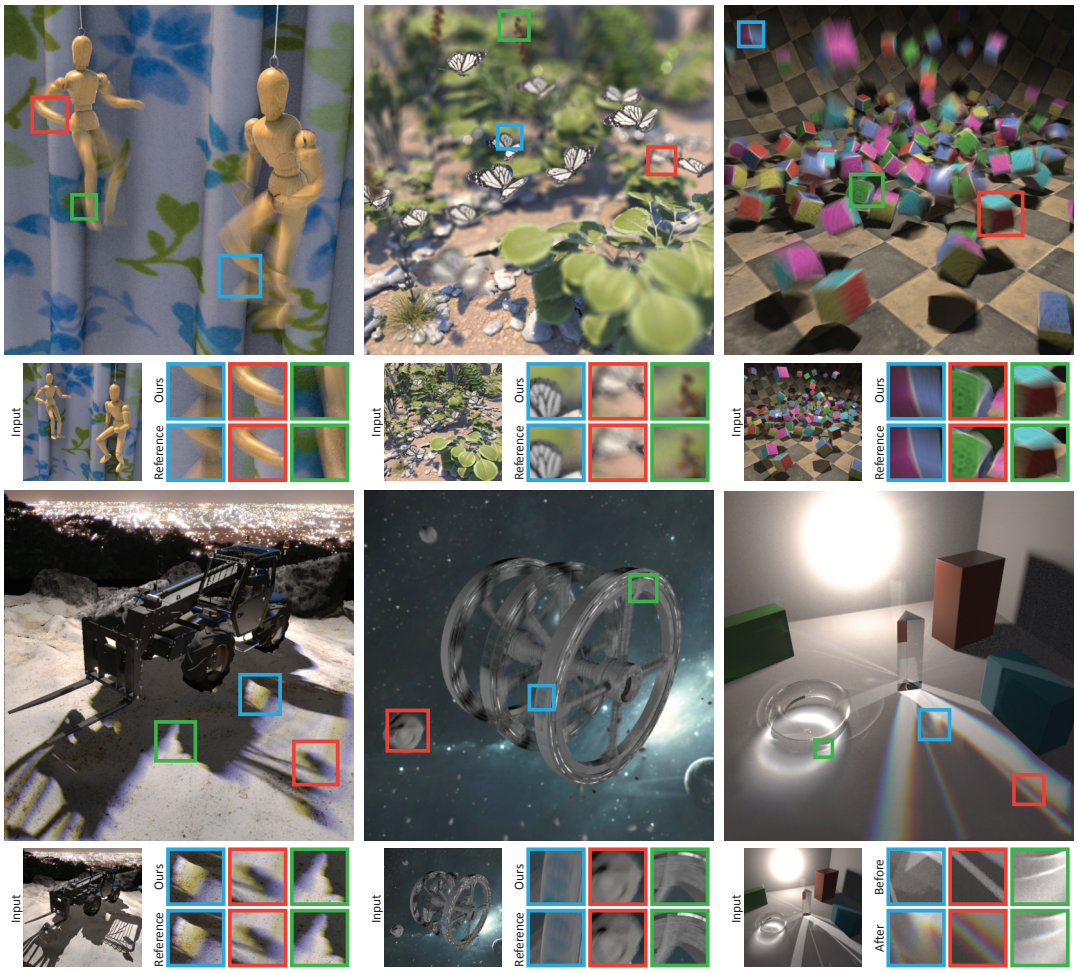


FIGURE 4.14: *Results of our method. The large figures show the end-result. Insets show comparisons between a multi-pass reference and ours.*

TABLE 4.2: *Timings for the individual stages. Two numbers in the* Layers *column denote the number of non-shadow and shadow layers. The timing of flow estimation for* Prism *does not include the spectral photon tracing, for which an external application was used. Timings given in italics are shown for completeness and are independent of our approach.*

Scene	Layers	n_d	n_s	Shading	Flow est.	Planning	Tiling	Warping	Aggreg.
Dancers	3	1	191	*115 s*	0.01 s	0.06 s	0.08 s	0.58 s	0.27 s
Garden	4	2	5385	*168 s*	0.05 s	0.91 s	0.09 s	30.96 s	9.15 s
Bouncing Cubes	5+5	1	848	*150 s*	0.05 s	1.15 s	0.09 s	4.89 s	1.36 s
Vehicle	2	2	11230	*95 s*	0.01 s	0.38 s	0.09 s	27.40 s	12.13 s
Space Station	5	3	53864	*426 s*	0.04 s	6.67 s	0.13 s	275.78 s	105.03 s
Prism	1	1	1059	*103 s*	0.01 s	0.12 s	0.08 s	3.26 s	0.74 s

TABLE 4.3: *Equal-time and equal-quality comparison to a path tracer.*

Scene	Equal Time (SSIM)	Equal Quality (Time)
Dancers	.70	3300 s
Garden	.38	4800 s
Bouncing Cubes	.80	480 s
Space Station	.89	1400 s

4.5.3 Discussion

Our cubature employs a regular sampling of the distribution domain. In theory, the approach therefore needs time exponential in the number of distribution dimensions n_d. In practice, the volume of all distribution flows (i. e., $n_s = \prod \mathbf{b}_i$) effectively constitutes the amount of work to be done. Consequently, it makes no difference if a combined lens-time sample space requires $10 \times 10 \times 10$ samples, or if motion blur alone smears a single pixel across 1000 pixels. We share this inherently strong dependency on the magnitude of the distribution effects to be produced with other rendering methods, including Monte Carlo approaches. Producing a novel view via minimal warping requires nothing more than two texture lookups in a 3×3 neighborhood per pixel. This is in contrast to producing a sample via ray-tracing, which exhibits larger constants and scales with geometric scene complexity, making cubature integration infeasible.

4.5.4 Limitations

Our main limitation, shared with typical assumptions in production, is to assume shading can be decoupled from resolving visibility. In several occasions such as small but strong highlights or shadows in the presence of motion blur, this assumption is violated and can become noticeable. The consequences of this decoupling have been analyzed and discussed elsewhere [Cook et al., 1987]. Another limitation is anti-aliasing: in our approach, every image pixel is associated with a single visibility sample and a unique depth value (modulo the layers of the LDI). Anti-aliasing can be applied by later super-sampling, but this remains costly as it increases both memory requirements and compute time.

While the limitations given above apply to all warping-based approaches, wiping is an error source unique to our approach. For flow configurations with complex visibility relations this results in image regions disappearing after occlusions for single branches of the sample tree. Wiping is rare in practice, not only because of the sample tree structure, but also because different geometry is placed on different LDI layers, which in turn are purposefully sorted to reduce the problem.

Our approach has the highest throughput if many very similar views are required. If the distribution space is not sampled densely (i. e., $p \gg 1$), searching becomes inefficient. Therefore, warping an image into another single image is better done using other methods.

4.6 Conclusion

In this chapter we have presented an approach to solve the problem of producing novel views that considers an entire family of images instead of individual single ones. We have shown that given the forward flow organized in a tree of flows, the inverse warping can become as simple as looking up a small pixel neighborhood for each pixel. Planning minimal warping provides a fast and flexible alternative to ray-tracing or multi-sampling using accumulation buffering, yet with enough flexibility to support all combinations of distribution effects such as motion blur, depth-of-field, soft shadows, and spectral shading. In particular, upcoming output devices such as high-refresh rate displays found in head-mounted displays or light field displays, require a massive amount of pixels in space and time. Yet, those images are redundant and almost identical. Minimal warping exploits this redundancy, resulting in flexible, efficient and high-quality imagery.

Chapter 5

Laplacian Kernel Splatting

5.1 Introduction

Depth-of-field (DoF) and motion blur (MB) are a key ingredient to the look and feel of most cinematic-quality feature films [Goy, 2013]. Reproducing them in synthesized imagery is a typical and well-understood part of most photo-realistic rendering systems. When it comes to efficient, interactive or even real-time rendering, current solutions to DoF and MB typically make several key assumptions that result in computational efficiency but come at the cost of reduced quality. A typical example is to assume DoF and MB to be independent, to be able to approximate the space-time lens transport by a convolution [Potmesil and Chakravarty, 1981] and often to approximate their reconstruction using Gaussian filtering [Munkberg et al., 2014; Vaidyanathan et al., 2015; Belcour et al., 2013; Egan et al., 2009; Soler et al., 2009]. In this work, we devise a method to synthesize or reconstruct cinematic quality motion blur and depth-of-field, while retaining most of the efficiency of typical approximations.

Input to our method are pixels (Fig. 5.1, a), which we see as light-field samples, labeled with additional geometric and dynamic information. We can work on pixels coming both from simple and layered images, as well as from a pinhole camera (synthesis), or stochastic path-tracing/rasterization (reconstruction). The point spread function (PSF) of each input pixel can affect a very large image area. Computing this contribution from each input point to a high number of output pixels is both the key to high quality, but regrettably also the reason for slow execution speed (Fig. 5.1, b).

Our key idea is to perform the required splatting operations in the Laplacian domain (Fig. 5.1, c). While the spatial extent affected by the typical PSF can be very large, it remains compressible, i. e., sparse, in the Laplacian domain (Fig. 5.1, Δ in b).

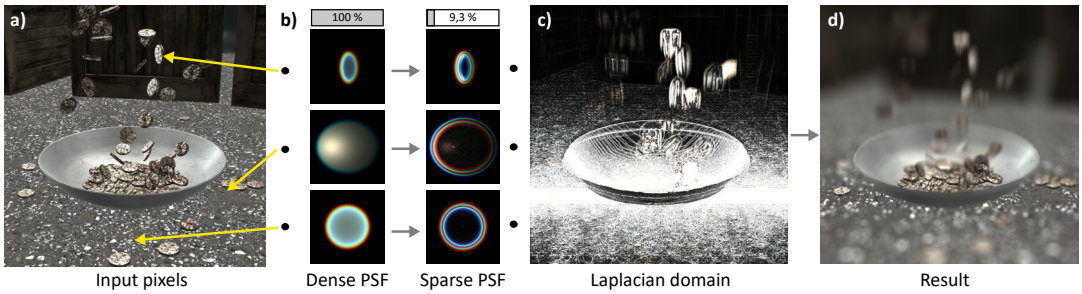


FIGURE 5.1: *Computing motion blur and depth-of-field by applying a point spread function (PSF) to every pixel* (a) *is computationally costly. We suggest splatting a pre-computed sparse approximation of the Laplacian of a PSF* (b) *to the Laplacian of an image* (c) *that under integration provides the same result* (d). *Note the circular bokeh combined with motion blur (1024×1024 pixels, 2 layers, 190 ms, Nvidia GTX 980Ti at .97 SSIM to a path-traced reference).*

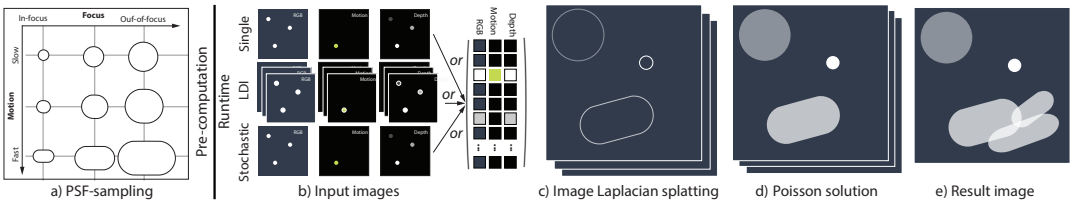


FIGURE 5.2: *Overview of our approach. The pre-computation* (a) *samples the space of all PSFs into a sparse representation. At runtime, one of the three types of images we support* (b) *are treated as lists of labeled pixels, here shown as three column vectors, the first holding appearance, the second motion and the third depth. To render, a PSF is splat for each pixel onto layered Laplacian images* (c) *that are integrated* (d) *and composed to produce the final result* (e).

Therefore, instead of splatting a dense contribution onto an image, we splat sparse points we call *spreadlets* onto the Laplacian of the image, which is finally transformed into the primal domain (Fig. 5.1, d) using a fast method [Farbman et al., 2011]. We operate on different models of spaces of all PSFs, depending on depth, motion and image position. A pre-process jointly optimizes for a sparse representation and a small reconstruction error of all PSFs in a particular space.

5.2 Overview

Our approach comprises of two steps: a pre-calculation (Fig. 5.2, a) followed by a runtime step (Fig. 5.2 b–e).

Pre-computation The pre-computation (Fig. 5.2, b and Fig. 5.4) samples the space of all PSFs according to a specific PSF model. Each PSF is converted into the Laplacian domain where it is approximated by a sparse set of optimized points. This "spreadlet" representation is stored on disk.

Input At runtime, input to our method are pixels labeled with shading and lens-time-etc. coordinates that we interpret as temporal light field samples. Such a sample captures the radiance emitted from a certain world position at a certain point in time through a certain lens position. This general notion allows to work on any simple, layered and stochastic (Fig. 5.2, b) image, produced either using a pinhole (OpenGL) rasterization, a deep framebuffer [Nalbach et al., 2014], LDI-style [Shade et al., 1998], using ray-tracing or by stochastic rasterization [Akenine-Möller et al., 2007].

Runtime Actual rendering is performed on (soft) global depth layers [Kraus and Strengert, 2007] (Fig. 5.2, c and Sec. 5.5). A global depth layer holds the appearance and transparency of pixels at a specific depth interval, where intervals are typically smaller close to the camera. Layering is required to capture non-linear occlusion relations while our splatting performs linear addition within a layer. Composing all layers provides the final image. For every input pixel, the pre-computed sparse PSF representation is drawn additively onto one or multiple layers, each holding the Laplacian of the image to reconstruct. When all pixels were splat, all layers are efficiently integrated [Farbman et al., 2011], i. e., converted from the Laplacian into the primal domain (Fig. 5.2, d). Finally, all layers are composed into the result image (Fig. 5.2, e).

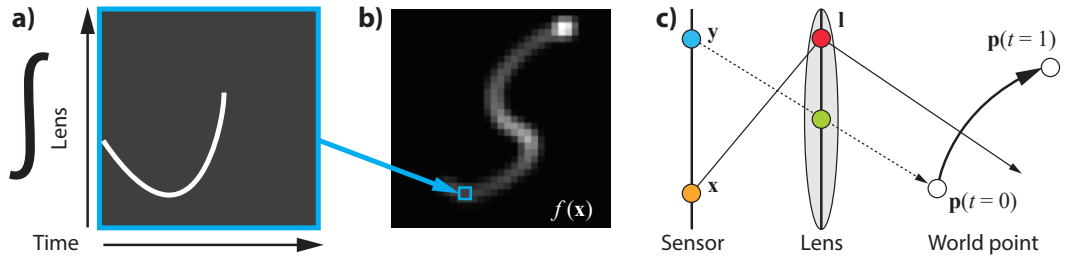


FIGURE 5.3: a) *The lens-time integration domain and range for a relative sensor location* $\mathbf{x}$. *At different time coordinates, different lens coordinates receive a contribution.* b) *There is one such function to integrate for every sensor location.* c) *The sensor-lens-world geometry.*

5.3 Background

Here we describe the formalization of DoF and MB into point-spread functions (Sec. 5.3.1) and derive the concept of differential rasterization (Sec. 5.3.2).

5.3.1 Point-spread Functions

Classic DoF can be described using point-spread functions (PSFs) [Kolb et al., 1995]. In this work we formalize the combination of DoF and MB using generalized *space-time* PSFs (Fig. 5.3). Such a PSF describes the contribution of a constantly light-emitting 3D point moving during a shutter interval $\mathcal{T}$ along a path $\mathbf{p}(t) \in \mathcal{T} \to \mathbb{R}^3$ to every relative sensor location $\mathbf{x}$. Therefore,

$$f(\mathbf{x}) = \int_{\mathcal{T}} \int_{\mathcal{L}} R(\mathbf{y} \to \mathbf{l}, \mathbf{p}(t)) \, \mathrm{d}\mathbf{l} \, \mathrm{d}t, \tag{5.1}$$

where, $\mathbf{x} \in \mathbb{R}^2$ is a 2D coordinate relative to $\mathbf{y} = P(\mathbf{p}(0)) + \mathbf{x}$, the absolute 2D sensor coordinate using projection P at start time, $\mathcal{L}$ is the lens area and $R(\mathbf{y} \to \mathbf{l}, \mathbf{p})$ is a Dirac that peaks when a ray $\mathbf{y} \to \mathbf{l}$ starting at 2D coordinate $\mathbf{y}$ in the sensor plane passing through 2D lens coordinate $\mathbf{l}$ (a two-plane light field parametrization) intersects the world point $\mathbf{p}$ (a "ray-point intersection").

As we will be dealing with multiple PSFs for different motion, lenses and absolute sensor locations we define a space of PSFs $f(\mathbf{x})(\mathbf{s})$ subject to a PSF parameter vector $\mathbf{s} \in \mathbb{R}^{n_s}$, as in

$$f(\mathbf{x})(\mathbf{s}) = \int_{\mathcal{T}} \int_{\mathcal{L}} R(\mathbf{s})(\mathbf{y} \to \mathbf{l}, \mathbf{p}(\mathbf{s})(t)) \, \mathrm{d}\mathbf{l} \, \mathrm{d}t,$$

so the ray formation R and the motion $\mathbf{p}$ depend on the PSF parameter vector $\mathbf{s}$.

5.3.2 Laplacian Rasterization

Our approach heavily relies on the fact that splatting can be performed in the Laplacian domain, i. e.,

$$\mathrm{splat}(f, I) = G(\Delta(\mathrm{splat}(f, I))) = G(\mathrm{splat}(\Delta f, I)),$$

where $\mathrm{splat}(f, I)$ denotes additive splatting (scattering) of a spatially-varying function f into an image I and G is Green's function (Sec. 2.1.2). The second equality holds due to the commutativity and distributivity of all involved operations. Please note that the above would also hold if splatting was replaced by convolution [Heckbert, 1986].

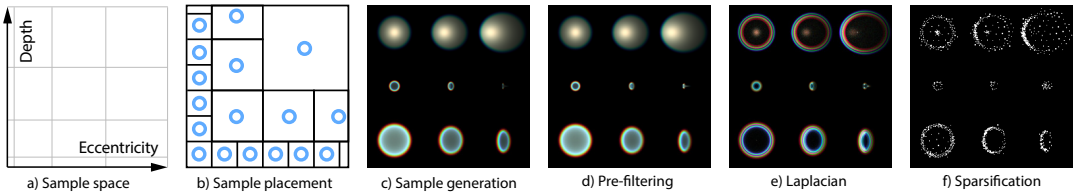


FIGURE 5.4: *The steps of our PSF sampling:* a) *Definition of the sample space.* b) *Non-uniform placement of samples (blue circles).* c) *Generation of the PSF at each sample.* d) *Pre-filtering in the sample space.* e) *Computing the Laplacian and* f) *sparsification into a set of points.*

We opt for splatting, however, as it delivers higher-quality results for our application domain.

To understand the relation of MB and DoF to the Laplacian, lets consider the cost of splatting PSFs onto pixels using different methods in the following three paragraphs:

Drawing a *solid circle* area appears to require filling all the, say, n_a pixels inside an image with n_p pixels. Simple drawing equals to evaluating a function $f(\mathbf{x})$ that is 1 inside the circle and 0 outside. Alternatively, we could draw $\Delta f(\mathbf{x})$ and later solve for f, leading to the same result. Now, drawing this Laplacian would in general also cost n_a drawing operations and additionally n_p operations to solve a Poisson problem $f = \mathsf{G}\Delta f$ using a pyramidal approach [Farbman et al., 2011]. This does not yet provide any benefit.

Consider drawing the *sum of n_p circles*, one for each pixel. This requires $n_\mathrm{a} \times n_\mathrm{p}$ fill operations. Drawing using the Laplacian, still requires $2 \times n_p \times n_a$ operations, while classic drawing requires $n_p \times n_a$ so no immediate benefit here either.

The key insight is that the Laplacian of the typical PSFs found in DoF/MB is very *sparse*: Splatting only a sparse approximation comprising of n_a pixels of the Laplacian, can result in a very similar reconstructed result. This means that n_a is much smaller for Δf than n_a is for f. Our approach builds on this property.

5.4 Pre-calculation: PSF Sampling

Sampling the space of all PSFs comprises of different stages (Fig. 5.4). First, we have to parametrize the space using a PSF model, such that we have a low-dimensional effective way to cover it as explained in Sec. 5.4.1. Second, we need to choose where to place samples, such that it is best represented where it is needed most (Sec. 5.4.2). Third, computing the PSF at specific sample positions can be challenging for complex lenses in combination with MB as explained in Sec. 5.4.3. Fourth, pre-filtering (Sec. 5.4.4) as with any sampling, also in the space of PSFs can prevent aliasing. Finally, the dense pre-filtered PSF sample is converted into a sparse set of points (spreadlets) approximating its Laplacian in an optimization step (Sec. 5.4.5).

5.4.1 PSF Model

Our approach allows for different PSF models (Tbl. 5.1) and resulting spaces to be discussed next in increasing complexity. Some PSFs are monochromatic, others support chromatic aberration. Coordinates in this space will be denoted as $\mathbf{s} \in \mathbb{S} = \mathbb{R}^{n_\mathrm{s}}$.

Some PSF models exhibit a natural rotational symmetry. As the Laplacian and, consequently, the integration operator G are rotational invariant, we can omit the

TABLE 5.1: *The PSF model zoo.*

Name	n_s	Color	Smp.	β	Size	Spar.
SIMPLELENS	1	✗	100	2.0	1.1 MB	31.3 %
COMBINED	2	✗	400	2.0, 1.0	4.2 MB	9.3 %
PHYSICALLENS	2	✓	4,000	2.5, 1.2	30.8 MB	2.2 %
VOLUME	1	✓	200	1.0	5.3 MB	18.4 %
STYLIZED	1	✓	200	1.5	9.0 MB	24.9 %

corresponding angular dimension during sampling. Note that this would not hold for other (differential) representations and reconstructions such as quad trees [Crow, 1984]. While an omitted sampling dimension significantly reduces memory, symmetries are not necessary for our approach to work.

Circular Depth-of-field The SIMPLELENS model is assuming a thin lens [Kolb et al., 1995]. The result is a one-dimensional space of monochromatic PSFs parametrized by the scalar circle of confusion (CoC) radius. The PSF is shift-invariant, i. e., points at the same depth are mapped to the same circle, regardless of the image position.

DoF and MB The COMBINED model adds motion blur to the previous model. It uses linear motion of constant projected speed and no motion in depth. This supports arbitrary viewer and object motion as well as deformation. More complex motion is to be composed from linear segments. This results in a two-dimensional monochromatic model, with scalar motion length (speed) as an additional parameter.

Physical Lens The PSF shape depends on absolute sensor position in our PHYSICALLENS model. Assuming this spatial layout to be rotation-invariant, we parametrize using the distance to the sensor center (eccentricity). This resulting space is two-dimensional: depth and eccentricity. Here, a chromatic PSF becomes important: In a physical lens model, light paths depend on wavelength and the PSF shows colored fringes. We use a simple bi-convex spherical lens in all our results.

Scattering Light scattering in participating media can also be described as a PSF [Premože et al., 2004]. Notably, the shape of the PSF in this case resembles the Green's function itself, i. e., it is highly compressible. For our VOLUME model we implemented volumetric light tracing in a homogeneous medium using Woodcock tracking and a Henyey-Greenstein (HG) phase function. The parameter of this space is the distance to the camera and the medium parameters remain fixed. We use a density of .9 and a HG anisotropy of .8 and RGB albedo of $(1, 1, .9)$.

Stylized Finally, we show how our approach is not limited to any physical model, but can use any mapping between pixel labels and PSFs. The STYLIZED PSF in our experiments is a logo with chromatic aberrations, scaled depending on the distance to the focal plane.

5.4.2 Sample Placement

Our approach achieves an appropriate cover of the PSF space using a *power remapping* and a *nested grid*. Both methods seek to place samples non-uniformly across the sample space, while retaining constant access time. For our PSF models it proved beneficial to allocate more samples to parameter ranges where artifacts due to quantization and pre-filtering (Sec. 5.4.4) would be most objectionable. This is typically the

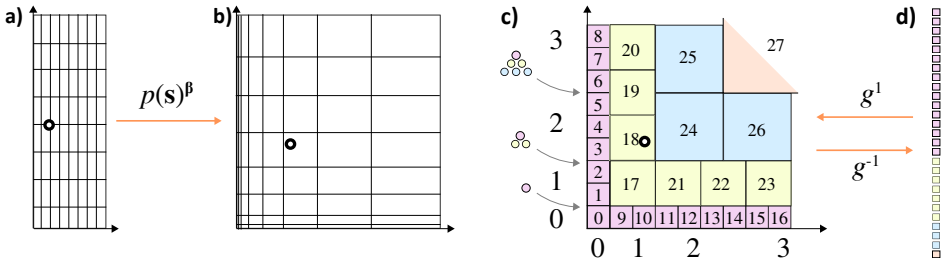


FIGURE 5.5: *Nested grid:* a) *Physical parameter grid.* b) *Its power remapping.* c) *Our nested grid topology.* d) *List of cells. (Please see text in Sec. 5.4.2.)*

case for PSFs where one or multiple coordinates are small, e. g., the slow-motion or near-focus PSFs in COMBINED.

Power Remapping The power remapping changes the physical PSF coordinates (Fig. 5.5, a) such that they cover the range from 0 to 1 non-uniformly by using a component-wise exponentiation $p(\mathbf{s}) = \mathbf{s}^{\beta}$ with a PSF model-specific vector β listed in Tbl. 5.1 as seen in Fig. 5.5, b. For $\beta > 1$ this results in a higher resolution for small coordinates and a lower resolution for large coordinates.

Nested Grid A straight-forward solution is to sample in a regular grid after the power-remapping. While a regular grid can be accessed in constant time it requires exponential pre-compute time and storage. Note that the power remapping does not change this property. We observe that our PSF models allow for a dramatically reduced resolution for large coordinates to such an extent that we chose to abandon the grid topology. Therefore, we suggest a *nested grid*, reducing the storage to polynomial time while retaining constant access time (Fig. 5.5, c).

We achieve this by increasing the length of the sample cell edges linearly with increasing coordinates for all required dimensions. This naturally results in a nested structure with different resolution levels (colors and large numbers in Fig. 5.5, c). We note that for the 2D 9×9 example shown in Fig. 5.5 a regular grid would have 81 entries, while our nested grid requires 28 entries. To work with such grids, we require two functions: a mapping $g(\mathbf{s}) \in \mathbb{R}^{n_s} \rightarrow \mathbb{N}^0$ from a continuous coordinate to an index and an inverse mapping $g^{-1}(i) \in \mathbb{N}^0 \rightarrow \mathbb{R}^{n_s}$ from an index to a coordinate. The forward map is required at runtime, the backward one at the pre-computation step.

The backward mapping $g^{-1}(i)$ is constructed incrementally by placing boxes until a level is filled, continuing until the space is filled. Note that some of those boxes are not cubic, as a cube would fall outside the space on such a lattice. It is consequently stored as a list of boxes (Fig. 5.5, d).

The forward mapping $g(\mathbf{s})$ is performed in two steps: First, we compute the minimum coordinate $s_{min} = \min(s_1, \ldots, s_{n_s})$. By construction of the nested grid, this value determines the resolution level. Since every level starts at a triangular number (triangles in Fig. 5.5, c), the level index l equals the *triangular root* of s_{min}, i. e., $l = \lfloor (\sqrt{8 s_{min} + 1} - 1)/2 \rfloor$. Second, the final cell index i is the sum $i = i_1 + i_2$ of the *inter-level* index and the *intra-level* index. The inter-level index i_1 is the sum of all indices before the current level l and we store this for all levels in a small table (indices of the boxes on the diagonal in Fig. 5.5, c). The intra-level index i_2 is computed just as in a regular grid within the level.

5.4.3 Sample Generation

Sampling $f(\mathbf{s}_i)$ for each $\mathbf{s}_i$ means to evaluate many (for each pixel) complex integrals as defined in Eq. 5.1. The value of the function is a 2D RGB image, in our case with a resolution of 512 pixels square. While a closed-form solution might exist for special models, such as SIMPLELENS, the task becomes harder for motion, forming capsule-shaped intensity-varying profiles as well as the famous cat eye-shaped flares. To compute the PSF of a complex lens system, including chromatic aberration is a research question on its own [Hullin et al., 2011]. The very general solution is light tracing [Dutré et al., 1993], which we opt to use, as it scales to complex lens systems including time-sampling. We typically use 700 million rays per PSF in a specialized GPU implementation. The high number of rays is required as differentiation in later steps will amplify any remaining variance. Note that our run-time efficiency is independent of the compute time of the PSF, only the sparsity in the Laplacian domain is relevant.

5.4.4 Pre-filtering

One sample $\mathbf{s}_i$ is a representative for an entire hyper-volume $\mathcal{S}_i$ in the space of PSFs. As we will use a single discrete pre-computed PSF sample that is nearest to the PSF required at runtime, a PSF sample should represent all PSFs that are closer to it than to any other. Failure to do so would result in aliasing or require a prohibitively large number of PSF samples. As a solution we suggest to pre-filter the PSF as in

$$\bar{f}(\mathbf{x})(\mathbf{s}_i) = \int_{\mathcal{N}(\mathbf{s}_i)} \int_{\mathcal{T}} \int_{\mathcal{L}} r(||\mathbf{s}_i - \mathbf{s}||) R(\mathbf{s})(\mathbf{y} \to \mathbf{l}, \mathbf{p}(\mathbf{s})(t)) \, \mathrm{d}\mathbf{l} \, \mathrm{d}t \, \mathrm{d}\mathbf{s} \,,$$

for all samples in its neighborhood $\mathcal{N}(\mathbf{s}_i)$, where $r(d)$ is a reconstruction kernel such as a Gaussian. This can be achieved by just another outer integration over the hyper-volume of the neighborhood in PSF space in the light tracing MC loop evaluating Eq. 5.1 above. Instead of tracing a particle through always the same PSF $f(\mathbf{s}_i)$, the PSF parameters are varied as well to fall into $\mathcal{N}(\mathbf{s}_i)$. For SIMPLELENS, instead of using a single discrete confusion, a range of confusions is used, etc. The neighborhood $\mathcal{N}(\mathbf{s}_i)$ is a simple-to-filter axis-aligned box with varying extent that can be computed from the inverse sample density used in Sec. 5.4.2.

Effectively, pre-filtering blurs the PSF spatially, trading aliasing against blur. As a typical PSF is spatially band-limited as well – no CoC in a real camera system is fully sharp – this appears plausible.

5.4.5 Sparsification

Instead of storing each PSF $\bar{f}(\mathbf{s}_i)$, which is dense, we store its sparse Laplacian $\Delta\bar{f}(\mathbf{s}_i)$. This helps representing entire areas of the dense PSF by sparse isolated peaks we call "spreadlets". We therefore would like to find a set of $n_{\mathrm{p},i}$ points with 2D position $\mathbf{x}_{i,j}$ and values $\Delta\bar{f}_{i,j}$ that minimizes the reconstruction cost

$$c(n_{\mathrm{p},i}, \mathbf{x}_i, \Delta\bar{f}_i | f) = \int_{(0,1)^2} | \underbrace{f(\mathbf{x})}_{\text{Signal}} - \underbrace{\mathsf{G} \sum_{j=0}^{n_{\mathrm{p},i}} \Delta\bar{f}_{i,j} \mathbb{1}(\mathbf{x}_i, \mathbf{x})}_{\text{Reconstruction}} | \mathrm{d}\mathbf{x},$$

in respect to a PSF f, where $\mathbb{1}(\mathbf{x}_0, \mathbf{x}_1)$ is an indicator function that is one if $\mathbf{x}_0 = \mathbf{x}_1$ and zero otherwise.

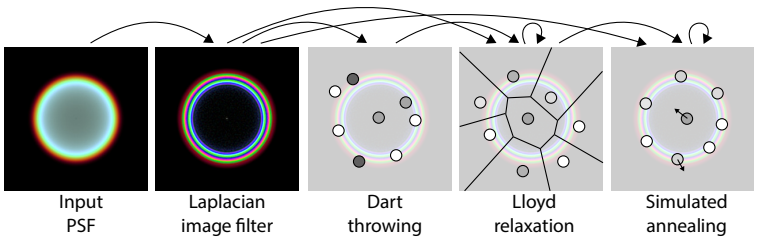


FIGURE 5.6: *Our four steps of PSF sparsification (conceptual illustration).*

Minimizing c poses several challenges: *i)* The cost landscape is highly non-convex, since every spreadlet adds one local extremum to it. *ii)* The reconstruction operator G has global support, resulting in a difficult condition. *iii)* The dimensionality of the problem is variable, since the point count $n_{\mathrm{p},i}$ is unknown. *iv)* We can splat only to discrete pixel coordinates, making the optimization a mixed problem where the $\mathbf{x}_i$ are integer and $\Delta\bar{f}_i$ are continuous.

Our attempts to partially (i. e., with a fixed $n_{\mathrm{p},i}$ and continuous $\mathbf{x}_i$) optimize via gradient descent or nonlinear conjugate gradient failed. However, we found the following practical procedure to minimize the cost in several steps (Fig. 5.6).

First, we apply a 3×3 Laplacian filter to $\bar{f}$, producing $\Delta\bar{f}$. This transformation into our target domain maps constant and linear regions to zero.

Second, we create a 2D Poisson disk pattern $\{\mathbf{x}_{i,0}, \ldots, \mathbf{x}_{i,n_{\mathrm{p},i}}\}$ with $|\Delta\bar{f}|$ as an importance function using a dart-throwing algorithm. We stop the placement after $10,000$ failed random attempts. This step initializes the sparse spreadlet representation we seek to obtain, by placing samples according to the local complexity of the PSF and at the same time determining the first free variable $n_{\mathrm{p},i}$ of our cost function.

To improve the spatial arrangement, we run 50 iterations of Lloyd relaxation [Lloyd, 1982], again using $|\Delta\bar{f}|$ for weighting.

Next, we sum the Laplacian PSF values in the Voronoi cell of every $\mathbf{x}_{i,j}$ and store it as a value $\Delta\bar{f}(\mathbf{x}_{i,j})$. This way, each Voronoi cell of the Laplacian is collapsed into a single pixel. This significantly increases sparsity, especially for areas with large cells. As the cell area is inversely proportional to the Laplacian PSF value, the values $\Delta\bar{f}(\mathbf{x}_{i,j})$ are very similar for different j, i. e., have a similar contribution to the final image.

Finally, we apply 400 steps of simulated annealing, where in each iteration, we first pick a fraction (1 %) of the integer positions and change them by at most one pixel, and second, re-assign the values $\Delta\bar{f}(\mathbf{x}_{i,j})$ according to the new Voronoi cells as described above. When two points happen to fall on the same integer grid coordinate, they can be merged, further increasing sparsity.

5.5 Runtime: PSF Splatting

The representation of all possible PSFs acquired in the previous section can now be used to efficiently compute new images with distribution effects. The procedure is similar to a trivial code that iterates all pixels and densely draws their PSFs in linear time [Lee et al., 2008]. Instead, we store (Sec. 5.5.1) and splat the sparse Laplacians of the PSFs (Sec. 5.5.2) of all pixels, followed by a final transform of the entire image from the Laplacian to the primal domain (Sec. 5.5.3).

5.5.1 Sample Storage

Each sample $\mathbf{s}_i$ has a varying number of points $n_{\mathrm{p},i}$. We concatenate all points $\mathbf{x}_{i,j}$ of all samples into a large sequence that is stored as a VBO P. The same is done for all function values $\Delta\bar{f}_{i,j}$ stored in a VBO V. The typical size of such a representation is several mega-bytes (Tbl. 5.1). The number of points changes for every sample: An in-focus sample requires fewer points than a moving lens flare. To efficiently handle a sequence of unstructured lists, we first pre-compute the vector of cumulative sums $n_{\mathrm{c},i}$ of all points in all samples with indices smaller than i and store it into a vector C.

5.5.2 Sample Splatting

Splatting happens for all input pixels in all layers independently and in parallel. We will therefore describe it for a single pixel at absolute sensor position $\mathbf{y}$ here. Let $\mathbf{s}$ be the PSF coordinate of that pixel. We now pick the sample $\mathbf{s}_i$ that is closest to $\mathbf{s}$ and draw all points to positions $\mathbf{y} + \mathbf{x}_i$ with value $\Delta\bar{f}(\mathbf{x}_i)$.

GPU Implementation Splatting is implemented in a compute shader that executes one thread for every pixel. Each thread fetches $\mathbf{s}$ for each pixel and computes the index i of the nearest PSF sample. After compensating for the non-linearities, the nested grid structure of our space (Sec. 5.4.2) makes this a simple and efficient $O(1)$ operation. If the PSF model employs symmetries, they have to be applied at this step: e. g., for motion blur, the spreadlets are pre-computed for motion in a certain reference direction and now have to be rotated to align with a specific direction of motion. Next the spreadlet is multiplied by the pixel color and drawn in a `for` loop over all points using `atomicFloatAdd` into four `R32F` textures.

Boundary The image has to be padded by a boundary large enough to accommodate for the largest PSF used. It is not sufficient to simply cut the kernel: consider a simple 1D example of a hat function that spans the image boundary. Depending on the PSF, that can include a translational part, to sample a certain output sensor size, the size of a virtual sensor that generates the input pixels might need to be substantially bigger or can be much smaller than the output sensor. The same applies for sampling considerations if the PSF is magnifying. Also note that the boundary only consumes memory, no splatting time and extra amount of integration time linear in its size.

Layering Details Since our approach operates on labeled pixel lists (Fig. 5.2, b), we naturally support (soft) global depth layers, LDIs [Shade et al., 1998], or deep framebuffers [Nalbach et al., 2014] as input formats. However, we need to splat into global depth layers for being able to properly pre-integrate per-layer radiance and opacity [Vaidyanathan et al., 2015]. Splatting is done independently for each output layer. If soft layering is desired [Kraus and Strengert, 2007], splats have to be drawn into more than one layer and weighted. In any case, we apply the re-weighting as suggested by Lee et al. [2008] when compositing the layers back-to-front.

In all our experiments we use n_{l} global input and output layers, where $n_{\mathrm{l}} = 1$ can be useful in some conditions. We bin them in units of constant parallax [Lee et al., 2009; Munkberg et al., 2014].

Note that layering only amplifies memory and merely shifts around the work: In particular the dominant splatting cost is not multiplied by n_{l} as a pixel is typically

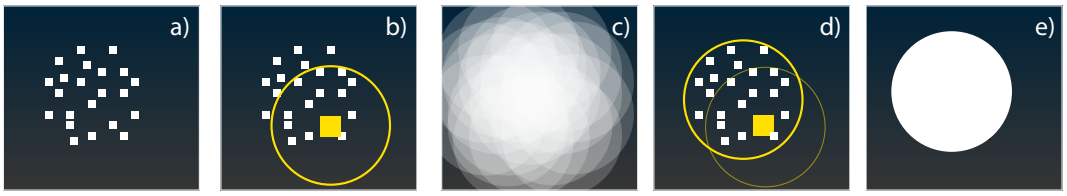


FIGURE 5.7: *Illustration of PSF splatting in a stochastic image with DoF:* a) *Stochastic image of a single bright point under defocus.* b) *A single PSF splat (yellow) centered around a single pixel at* $\mathbf{y}$. c) *Overlay of all PSFs.* d) *The same single splat, but now centered around* $\mathbf{y}'$. e) *Overlay of all PSFs.*

only contained in one layer (or two layers if soft) and empty pixels will be culled very early on.

Stochastic Frame-buffers Special considerations are to be taken if the framebuffer is stochastic (Fig. 5.7, a). An example is DoF: a surface projecting to the sensor position $\mathbf{y}'$ at $\mathbf{l} = t = 0$ will move to a new sensor position $\mathbf{y}$ (yellow point in Fig. 5.7, b), as they are distributed across the entire circle of confusion.

Just running the above procedure on this data would mean to place another circle of confusion on an already distributed pixel i. e., to apply the PSF twice (Fig. 5.7, c). As every pixel has a unique distribution coordinate $\mathbf{s}_i$, we can re-compute its original absolute sensor position $\mathbf{y}'$, and splat the PSF around $\mathbf{y}'$ instead of $\mathbf{y}$. Conceptually, for the DoF example, this realigns the PSF such that it is drawn around the center of the CoC it belongs to, not around the pixel itself that is part of the CoC (Fig. 5.7, d–e). Consequently, non-stochastic input is a special case of stochastic input with $\mathbf{y} = \mathbf{y}'$.

5.5.3 Integration

After all points for all pixels were drawn, the image is transformed from the Laplacian into the primary domain. This is efficiently done using convolutional pyramids [Farbman et al., 2011] which takes 6 ms for a 1024×1024 image.

5.5.4 Fast Track

In practice, some PSFs can have less sparsity than others. The main speed-up we achieve is for large PSFs that are sparse, which also implies that splatting small PSFs using our sparsification scheme is not effective. Fortunately, our approach can combine both strategies seamlessly. To this end, we maintain two images per layer: A Laplacian image to which sparse points are splatted and a direct one. The decision to draw sparse or dense is made simply on the number of points. Both images are added after the Laplacian image was integrated. This strategy is used in all results shown in this chapter and typically amounts to about 9% of the PSFs.

5.6 Results and Discussion

In this section we show qualitative (Sec. 5.6.1), quantitative (Sec. 5.6.2) results of our approach and an analysis of its properties (Sec. 5.6.3). We compare OUR approach to different alternatives:

A SPLATTING approach draws the dense ground-truth PSF. This is an upper bound on what we can achieve, as our Laplacians are just an approximation. We would hope to achieve similar quality, just at a much higher speed.

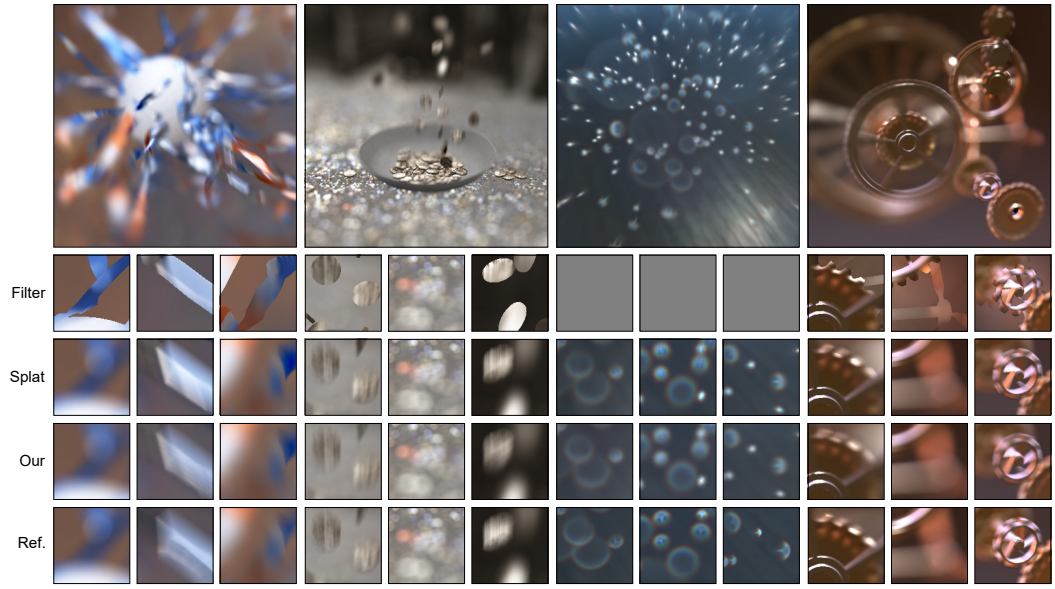


FIGURE 5.8: *Results of our (large) as well as other (insets) synthesis approaches on different scenes. (1024×1024 pixels).*

The FILTERING method uses the same space of PSFs we use, but instead of splatting the PSF, we filter using the PSF. Note that this is an upper bound on what any filtering-based method can achieve. We expect to achieve both higher quality and speed. We do not apply filtering to PHYSICALLENS, as neither the pre-computed dense PSF images fit into memory, nor is it feasible to compute them for every pixel on the fly.

Many methods to remove noise, in particular the noise specific to path-traced images, exist [McCool, 1999; Kontkanen et al., 2006; Sen and Darabi, 2012; Egan et al., 2009; Soler et al., 2009; Belcour et al., 2013; Munkberg et al., 2014]. As all these methods have different trade-offs and assumptions, we here opt for BM3D, a general state-of-the-art image denoiser [Dabov et al., 2006] that has been used to denoise path-traced images before [Kalantari and Sen, 2013].

The REFERENCE method uses path tracing based on a reasonably implemented GPU ray-tracer with an SAH-build BVH where nodes are extended to bound space-time primitives.

5.6.1 Qualitative Results

Qualitative results of synthesis and reconstruction are shown in Fig. 5.8 and Fig. 5.9.

Synthesis In Fig. 5.8 we see in "Whirl" how OUR method produces detailed PSFs that add cinematic quality to the shot, with circular bokeh and long motion trails. MB and DoF arising from the complex motion patterns are faithfully synthesized, while the colorful specular highlights in "Coins" are transformed into overlapping, yet distinct circles of confusion. The rotational motion in "Gears", here in combination with bright specular highlights under defocus, gives rise to appealing high-contrast image regions entirely defined by the PSFs produced. The "Rain" scene rendered with the physical model shows the expected complex cat-eye shapes in image corners, where the CoC is deformed. Chromatic aberration is reproduced as well. The same figure shows the comparison to alternative methods as insets. We see that the

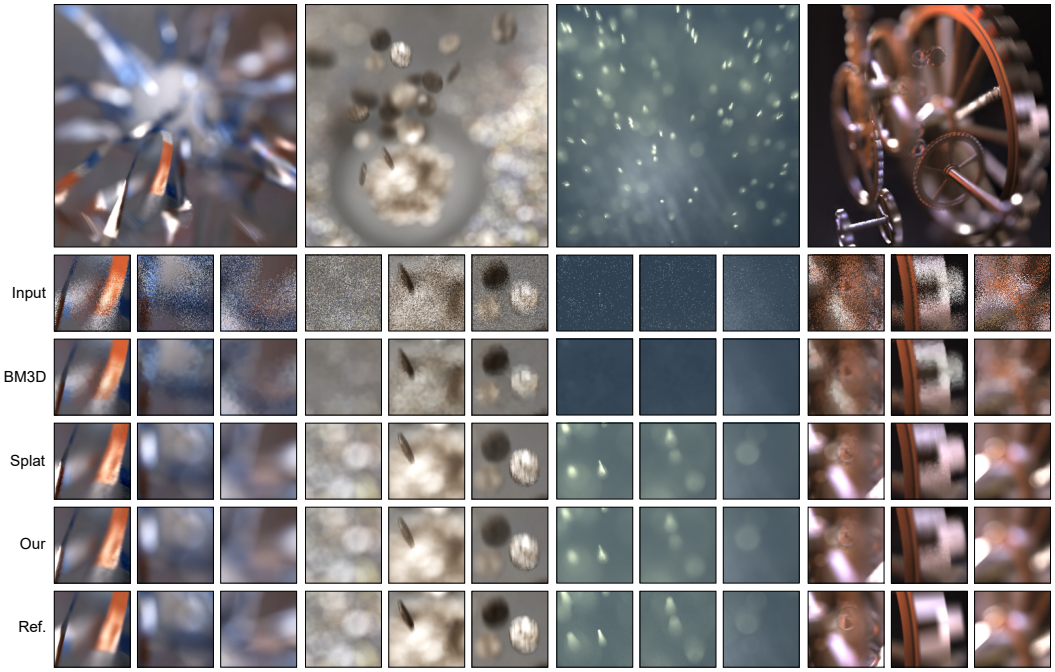


FIGURE 5.9: *Results of our (large) as well as other (insets) reconstruction approaches on different scenes. (1024×1024 pixels, 1spp path traced input).*

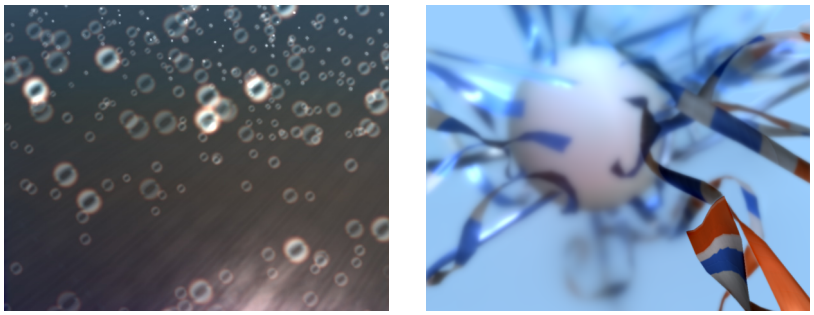


FIGURE 5.10: *Results for the* STYLIZED *(left) and* VOLUME *(right) PSF models.*

FILTERING method looks quite different, while the method based on SPLATTING is very similar to OURS and the REFERENCE.

Results for the STYLIZED and VOLUME PSF models are shown in Fig. 5.10. We see how stylization provides a non-physical effect where the PSFs take the shape of a logo, while for the participating media PSF the colors shift according to the model. In addition, a distinct blur can be observed, in particular for locations in the background, as one would expect from scattered light.

Reconstruction In Fig. 5.9 we use our method for reconstructing from stochastic input. While the input already contains MB and DoF, our method preserves it, yet removes the noise. In several cases, our method reconstructs features that are almost invisible in the input to the naked eye, such as in "Rain". Please note how our carefully aligned high-frequency PSFs are able to reconstruct subtle semi-transparencies.

5.6.2 Quantitative Results

We provide quantitative results in terms of comparison to a reference and alternative approaches. Quality is measured in SSIM (larger is better) and speed in milliseconds. All comparisons are done in resolution 1024×1024 on an Intel Xeon E5-1607 CPU in combination with a Nvidia Geforce GTX 980Ti GPU. The pre-computation requires roughly 20 seconds for one PSF. Numerical results are stated in Tbl. 5.2 for synthesis and in Tbl. 5.3 for reconstruction. We see our approach consistently has the highest speed and provides images of high similarity to the reference. Splatting has a similar error, but is typically slower by almost one order of magnitude, as our representation is typically one order of magnitude more sparse.

TABLE 5.2: *Numerical results for Fig. 5.8.*

		OUR		FILTER		SPLAT		REF.
	Model	Time	Err.	Time	Err.	Time	Err.	Time
Whirl	Comb.	**88.0** ms	**.94**	299.6 ms	.81	722.8 ms	.94	592 s
Coins	Comb.	**163.0** ms	**.96**	368.1 ms	.92	1930.9 ms	.96	232 s
Rain	Phy.	**126.3** ms	**.90**	—	—	7170.4 ms	.88	>1000 s
Gear	Comb.	**65.9** ms	**.94**	186.2 ms	.87	783.8 ms	.96	>1000 s

TABLE 5.3: *Numerical results for Fig. 5.9.*

		OUR		BM3D	SPLAT		REF.
	Model	Time	Err.	Err.	Time	Err.	Time
Whirl	Comb.	**75.8** ms	**.95**	.93	605.3 ms	.95	346 s
Coins	Comb.	**496.4** ms	**.94**	.91	2652.3 ms	.94	366 s
Rain	Comb.	**499.5** ms	**.96**	.89	2276.2 ms	.96	>1000 s
Gear	Comb.	**1699.1** ms	**.91**	.91	3495.8 ms	.92	>1000 s

5.6.3 Analysis

Here we analyze how our approach and variants thereof perform under different conditions. As we already established we can generate images similar to a reference, provided we have a suitable (sparse and low-error) PSF representation, we perform analysis purely on the space of PSFs images (the test set) instead of complex images. Three qualities are important: i) the *sparsity* in percentage (which translates into computational efficiency, up to a small additive constant overhead for integration), ii) the *similarity* measured between PSFs in terms of SSIM, and finally, iii) a good *efficiency* ratio between the two, measured in similarity-per-sparsity.

Sparsity Our representation achieves an average sparsity of 9.3 % (Fig. 5.11, a) while retaining an average similarity of .97, which results in an efficiency if 10.75. We further look into the distribution of sparsity (Fig. 5.11, b) and similarity (Fig. 5.11, c). The sparsity distribution is peaked around very sparse PSFs and most solutions have a high similarity. Only very few PSFs have a low similarity.

Laplacian Domain We have chosen to use the Laplacian domain while other representations could also achieve the sparsification required. A typical sparse coding that is efficient to produce and generate is wavelets, i. e., a quad tree [Crow, 1984] (QT). To this end we convert the test set into the QT domain. In Fig. 5.11, a, adjusting QT for a

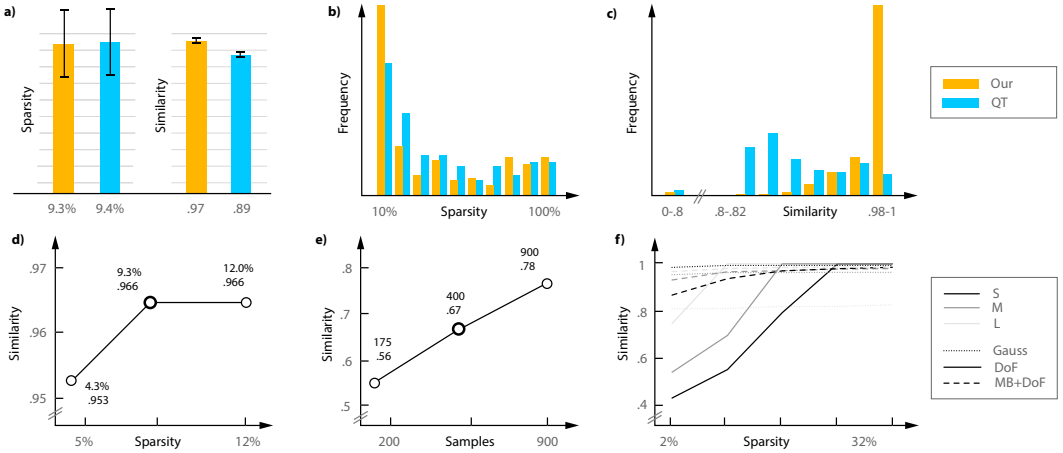


FIGURE 5.11: *Analysis for* COMBINED*:* a) *Mean sparsity and similarity for our approach and a quad tree (QT).* b, c) *Sparsity resp. similarity histograms for Laplacian and QT. Frequency counts how often a PSF with this property occurs.* ***d)*** *Relation of sparsity and similarity for a 0.5× and a 2×-sparsity operational point.* e) *Relation of sample count and similarity for the same operational points.* f) *Relation of sparsity (log. scale) and similarity for differently-sized (S, M, L) PSFs.*

similar sparsity, we find a lower similarity of .89, providing an inferior efficiency of 9.67. A distribution of sparsity and similarity is seen in Fig. 5.11, b and c. We see that the sparsity is not as peaked for very sparse solutions while at the same time, many more solutions are of low similarity. Note that a QT is not rotation-invariant, leading to an order of magnitude more memory requirement. This indicates the Laplacian is a good representation for PSF sparsification in terms of memory and speed.

Spreadlet Count We have chosen two other operational points of our approach in Fig. 5.11, d where the average sparsity is roughly half and twice as large as the one we suggest to use. We observe that quality saturates at a very high similarity to a reference, indicating that sparsity can be used to control quality.

Sample Count We instrument the relation of PSF sample count and average similarity between the closest sampled PSF and the ground-truth PSF at 1,000 random coordinates in Fig. 5.11, e. We see that increasing the number of samples increases similarity. This is because pre-filtering introduces blur. In practice, the PSF is not applied to individual pixels, but to groups of pixels which also results in blurring, resulting in the good end-image similarity we observe.

Approximation Quality Our approach builds on the observation that a sparse approximation of the Laplacian of certain PSFs is feasible. To better understand the approximation behavior of our sparsification scheme we analyze the reconstruction quality for different PSF types of different sizes (S/M/L) with varying sparsity in Fig. 5.11, f. We compare isotropic Gaussian functions ($\sigma = 25/40/55\,px$) with our DoF CoCs ($r = 25/50/100\,px$) and capsule-shaped motion-blurred CoCs ($r = 50/50/100\,px, l = 50/128/128\,px$). We observe that the reconstruction quality for the typical PSFs we target increases with spatial extent, while the opposite can be found for corresponding Gaussians. This can be attributed to the relatively small total variation of our large-scale target PSFs, allowing a representation by a small set of Laplacian peaks. This is in contrast to the uniform smoothness of Gaussians. For

FIGURE 5.12: *Comparing the effect of pre-filtering for the* COMBINED *model.*

the binary DoF kernels, which exhibit natural sparsity in the Laplacian domain, the reconstruction quality saturates at 100% once enough spreadlets are allocated.

Optimization Here we report results of ablational studies regarding our optimization procedure. The mean L1 error across our PSF corpus relative to dart throwing only (100%) is 96.7% for dart throwing and Lloyd relaxation, 87.1% for dart throwing and simulated annealing, and 86.5% for the full procedure. While this is a modest mean improvement, it removes outliers that are visually disturbing.

5.6.4 Limitations

While our approach is impaired by typical limitations inherent to image-based synthesis and reconstruction, such as layer quantization and the assumption of diffuse reflectance, we discuss three limitations and artifacts unique to our approach in the following paragraphs.

Pre-filtering The effect of pre-filtering for the COMBINED model is studied in Fig. 5.12. Without pre-filtering, we see discontinuity errors. With pre-filtering the aliasing is converted into blur, a less suspicious artifact when comparing to the reference.

Spreadlet Undersampling Occasionally, for PSFs with small spatial extent and high frequencies, the dart throwing step of our optimization procedure fails to allocate enough spreadlets. This results in blotchy artifacts after reconstruction, as can be observed in the third main column of Fig. 5.8.

Curse of Dimensionality Our approach requires full sample coverage of the PSF spaces it operates on. Even though the sample count is reduced by utilizing our nested grid structure and by exploiting rotational symmetries, higher-dimensional PSF models, like a physical lens in combination with a higher-order motion model, would require a prohibitive amount of pre-computation.

5.7 Conclusion

We have described a method to achieve interactive performance when synthesizing and reconstructing combinations of DoF and MB. The key observation of this chapter is that many PSFs are sparse in the Laplacian domain. Our method gains its efficiency by computing a sparse approximation of a multitude of PSFs in a pre-process, which enables cinematic-quality distribution effects with a run-time performance that is an order of magnitude faster than previous methods.

Chapter 6

Perceptual Real-time 2D-to-3D Conversion Using Cue Fusion

6.1 Introduction

The majority of images and videos available is 2D and automatic conversion to 3D is a long-standing challenge [Zhang et al., 2011]. For applications such as view synthesis, for surveillance, autonomous driving, human body tracking, relighting or fabrication, accurate physical depth is mandatory, and obviously binocular disparity can be computed from such data, resulting in a perfect stereo image pair. However, for 2D-to-3D stereo conversion, such physical depth is not required. Instead, we seek to compute perceptually plausible disparity in this work. It differs from physical depth by three properties. First, the absolute scale of disparity is not relevant, and any reasonable smooth remapping [Lang et al., 2010; Didyk et al., 2012] is perceived equally plausible and may even be preferred in terms of viewing comfort and realism. Second, the natural statistics of depth and luminance indicate that depth is typically spatially smooth, except at luminance discontinuities [Yang and Purves, 2003; Merkle et al., 2009]. Therefore, not reproducing disparity details can be acceptable and is often not even perceived, except at luminance edges [Kane et al., 2014]. Third, the temporal perception of disparity allows for a temporally coarse solution, as fine temporal variations of disparity are not perceivable [Howard and Rogers, 2012; Kane et al., 2014]. Consequently, as long as the error is 2D-motion compensated, depth from one point in time can be used to replace depth at a different, nearby point in time.

Our method is modular (Sec. 6.2) and based on priors learned in a pre-process (Sec. 6.3) combined with stereo cues extracted from 2D images or videos at runtime (Sec. 6.4). Both priors and cues are represented as normal distributions allowing to fuse a plausible disparity map with high spatial and temporal resolution in real-time (Sec. 6.5). Image-based rendering produces a stereo video stream from this map (Sec. 6.6). In Sec. 6.7 we validate our notion of perceptually plausible disparity and discuss our results. We find that our system can perform 2D-to-3D conversion at ca. 35 Hz for HD video and compares favorable to off-line methods in terms of different error metrics as well as user ratings. In summary, our contributions are *(i)* a real-time 2D-to-3D conversion system based on the fusion of learned priors and depth cues into a coherent disparity estimate, *(ii)* an analysis of the importance of different depth cues in different scenes based on estimated confidence, and *(iii)* a perceptual analysis of disparity plausibility, including spatial and temporal sampling requirements for perceptual disparity processing tasks.

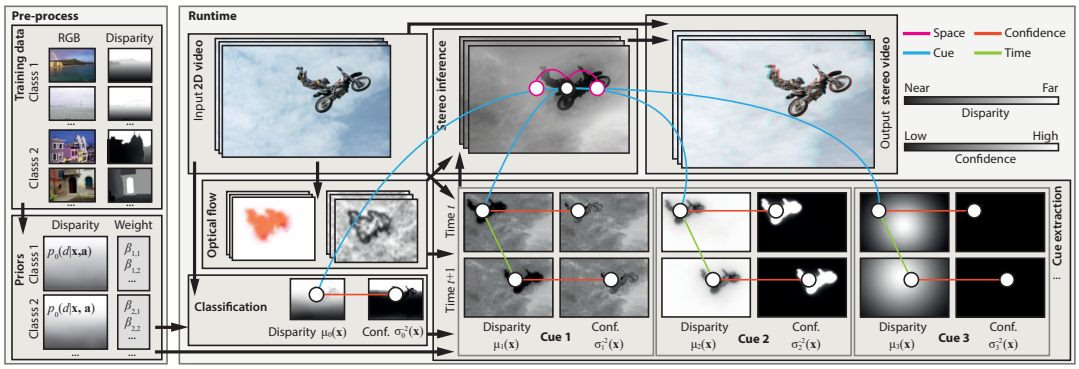


FIGURE 6.1: *Overview of our approach* (from left to right) *as described in Sec. 6.2. The grey coding used is annotated in the top right.*

6.2 Overview

An overview of our approach is shown in Fig. 6.1. It has two main parts: a pre-process (Sec. 6.3) to extract disparity priors (Fig. 6.1, left) and a runtime component (Fig. 6.1, right). While the pre-process uses many example images and requires considerable time, the runtime components execute in real time.

At runtime, first disparity and disparity confidence maps are extracted from monocular images (Sec. 6.4). This is the most computationally intensive part of our pipeline and implemented as parallel algorithms to require only a few milliseconds each. We support a flexible combination of both static cues (defocus, aerial perspective, vanishing points and occlusions) and dynamic cues (depth-from-motion). Each cue alone often has a low confidence in many areas and might contradict other cues. The cue evidence is then fused into plausible disparity maps (Sec. 6.5) using a robust maximum a posteriori (MAP) estimate [Knill and Richards, 1996]. This fusion happens again in real time, producing results that are smooth in time and space, except at luminance edges. Finally, the monocular input image is converted into a stereo image pair obeying the disparity gradient limit (Sec. 6.6).

We will use a simplified disparity space ranging from 0 (close, depicted as black) to 1 (far, shown as white). As our goal is producing plausible disparity, we choose *not* to work in physical units like, e. g., the difference of vergence angles [Howard and Rogers, 2012] or pixel disparities [Szeliski, 2010]. Our perceptually plausible disparities arise instead from a smooth and monotonic remapping of physical disparities and are inspired by the way depth maps for manual stereoscopic conversion are painted. The perceptual effect of monotonic remappings of disparity is analysed in Sec. 6.7.2. The resulting disparity values will later be remapped to a comfortable range depending on the reproduction device.

6.3 Pre-processing

In a pre-process we learn prior information about disparity for certain classes of images and how to detect those classes.

6.3.1 Disparity Priors

Priors model what is known about disparity in general without considering any specific image. This information is acquired from example depth images, validated and calibrated, and finally fit to a conditional distribution.

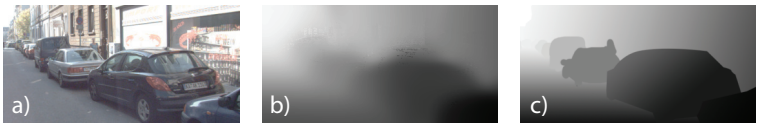


FIGURE 6.2: *Example disparity maps for the scene class "street".* (a) *Appearance.* (b) *Disparity from sensor.* (c) *Disparity from human annotation.*

A disparity prior is the probability distribution of disparity $p_0(d)$. For efficient storage and computation, the probability distribution $p_0(d) = \mathcal{N}(d|\mu_0, \sigma_0)$ is modeled as a normal distribution $\mathcal{N}$ of a certain mean μ_0, standard deviation σ_0, and variance σ_0^2 in this work. Furthermore, our priors $p_0(d|c, \mathbf{x}, \mathbf{a})$ are *conditioned* on three parameters: the scene class c (the depth distribution in "street" is different from "open countries"), the location $\mathbf{x} \in \mathbb{R}^2$ inside the image (the upper areas are more likely to be distant) and the appearance (RGB color) $\mathbf{a} \in \mathbb{R}^3$ (blue in the top of a forest image is more likely distant than green). For final cue fusion, scene class, image location and appearance are known and *unconditioned* priors will be used. Formally, the conditioned prior is defined as two 6D maps containing mean disparity $\bar{\mu}_0(c, \mathbf{x}, \mathbf{a})$ and the confidence of disparity $\bar{\sigma}_0^{-2}(c, \mathbf{x}, \mathbf{a})$. A high-variance value is found for a wide and unreliable distribution, while a high-confidence value $\bar{\sigma}_0^{-2}$ indicates a reliable estimate.

We use 10 representative scene classes consisting of about 40 example images each. Disparity maps were acquired both by sensors and by human annotation. Sensor-acquired classes are "street" and "indoor". For all other classes ("close-up", "coast", "forest", "inside city", "mountain", "open country", "portrait", "tall buildings"), depth maps were painted manually. Annotation was done in parts by 2D-to-3D conversion professionals, and experienced users of image manipulation software. Images have a resolution of ca. 100 k pixels.

To compare human annotation performance to physical measurements, additional manual depth map painting was repeated for classes where sensor measurements are available by participants naïve with respect to the purpose of the procedure. A linear fit from painted depth x to physical vergence angles y with $y = .74\,x - .03$ has an error of adjusted $R^2 = .40$, indicating humans do a fair job when painting vergence compared to a sensor (Fig. 6.2).

Priors are extracted from example data independently for each class (see examples in Fig. 6.3). Each prior is represented as a 5D regular grid where the spatial dimension is discretized into 62×38 and the color dimension into $3 \times 3 \times 3$ bins. Normalized image coordinates between 0 and 1 are used for the spatial component and $YCrCb$ color coordinates for the color component. Consequently, our prior contains $n_\mathrm{b} = 63\,612$ bins, with coordinates denoted as $\mathbf{b}_i \in \mathbb{R}^5$. The 2D positions and 3D colors of the n_s input pixels from all input images from that class are concatenated into a set of 5D samples $\mathbf{s}_j \in \mathbb{R}^5$, where each sample is labeled with its disparity d_j. Note that the number of bins is much smaller than the number of samples, $n_\mathrm{b} \ll n_\mathrm{s}$ (Fig. 6.4). Prior mean and confidence are each computed independently for all grid cells in two consecutive passes. In the first pass, the prior mean is computed as

$$\bar{\mu}_{0,i} = \frac{\sum_{j=1}^{n_\mathrm{s}} w_{ij} d_j}{\sum_{j=1}^{n_\mathrm{s}} w_{ij}} \quad \text{where} \quad w_{ij} = \alpha_i \psi(\mathbf{s}_j, \mathbf{b}_i) \tag{6.1}$$

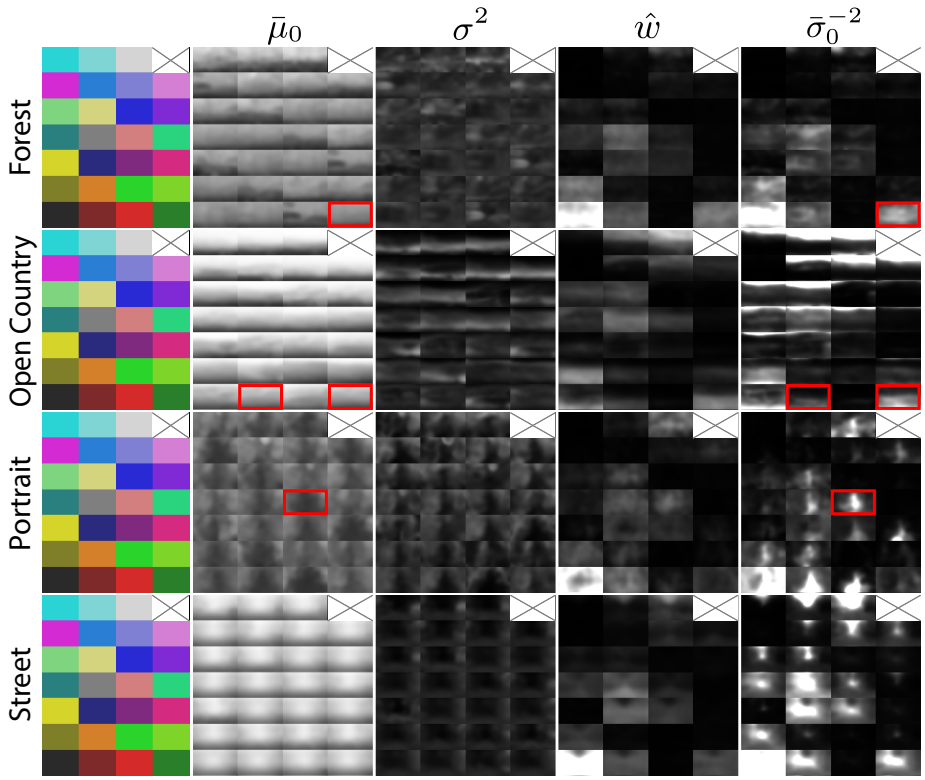


FIGURE 6.3: *Mean, variance, weight and confidence* (columns) *at different colors* (tiles) *for priors of different classes* (rows). *For "forest", green central pixels have a medium depth. For "open country", brown and green lower pixels have a nearby depth. For "portrait", skin-colored central pixels are more nearby.*

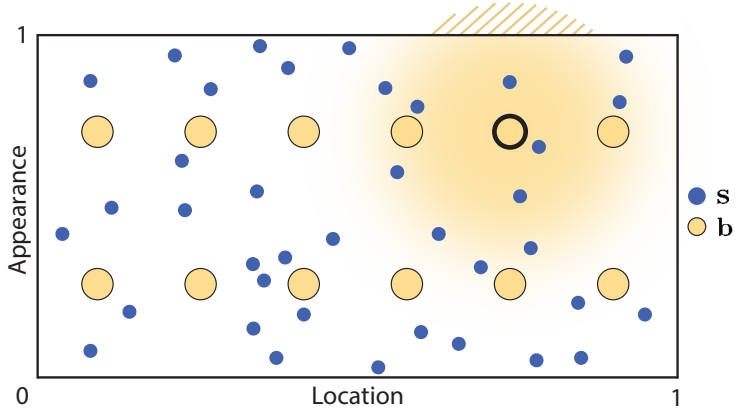


FIGURE 6.4: *A schematic of our prior extraction procedure with a 1D location and appearance domain. The samples* $\mathbf{s}_j$ (blue dots) *stem from the example images. The sparse bins* $\mathbf{b}_i$ (orange circles) *constitute the actual prior data* ($\bar{\mu}_{0,i}$ *and* $\bar{\sigma}_{0,i}^{-2}$) *to be determined. This is done by calculating the Gaussian weight* $\psi(\mathbf{s}, \mathbf{b})$ *for each possible sample/bin combination* (shown as a radial gradient for the highlighted bin). *To avoid a boundary bias* (hatched) *a correction* α_i *has to be determined for each bin.*

and $\psi(\mathbf{s}, \mathbf{b}) = \exp\big(-(\mathbf{s}-\mathbf{b})^\mathsf{T}\mathsf{A}(\mathbf{s}-\mathbf{b})\big)$ is a Gaussian kernel with a diagonal precision matrix A. For all results in this chapter, the empirically chosen matrix entries are $\mathsf{A}_{11} = \mathsf{A}_{22} = 75$ for the spatial and $\mathsf{A}_{33} = \mathsf{A}_{44} = \mathsf{A}_{55} = 40$ for the appearance term. The normalization α_i for bin i is required because the 5D population can be highly non-uniform, and we use Gaussian filters of infinite support instead of compact (e. g., Epanechnikov) kernels. At the same time, our number of bins introduces a boundary bias for bins closer to the surface of the space-appearance cube which would receive a lower total weight compared to other pixels. To compensate for this effect, we compute a correction

$$\alpha_i = \left(\int_{(0,1)^5} \psi(\mathbf{s}, \mathbf{b}_i) \mathrm{d}\mathbf{s} \right)^{-1}$$

of each 5D bin using Monte Carlo integration and normalize the result of each bin by this value. In the next pass, prior per-bin variance and weight

$$\sigma_i^2 = \frac{\sum_{j=1}^{n_\mathrm{s}} w_{ij} (\bar{\mu}_{0,i} - d_j)^2}{\sum_{j=1}^{n_\mathrm{s}} w_{ij} - \frac{\sum_{j=1}^{n_\mathrm{s}} w_{ij}^2}{\sum_{j=1}^{n_\mathrm{s}} w_{ij}}} \quad \text{and} \quad \hat{w}_i = \frac{\sum_{j=1}^{n_\mathrm{s}} w_{ij}}{\sum_{i=1}^{n_\mathrm{b}} \sum_{j=1}^{n_\mathrm{s}} w_{ij}}$$

are computed. The final prior confidence is

$$\bar{\sigma}_{0,i}^{-2} = \frac{\hat{w}_i}{\sigma_i^2}. \tag{6.2}$$

For the confidence $\bar{\sigma}_{0,i}^{-2}$ of the prior to be high, the variance σ_i^2 has to be low (agreement of samples to the mean) and the weight $\hat{w}_i$ has to be high (many samples similar to this bin). This combination prevents bins with a low number of samples to have a high confidence just because their estimate of variance is not stable.

6.3.2 Scene Classification

Priors depend on the scene class c which is found from the monocular input RGB image. To this end, an image classifier is trained from example images that were manually labeled by their scene class. To meet our real-time requirements at test time and following ideas from Torralba [2009], the image downscaled to 8×8 pixels is used as a feature vector. A linear Support Vector Machine is trained using gradient descent to separate each class from the other classes (one-versus-one). At test time, we count the number of wins for each class over the other classes and pick the class c with the largest number of wins.

6.4 Depth Cues

We model the i-th depth cue as a conditional probability distribution $p_i(d|\mathbf{x})$ of disparity d given a position $\mathbf{x}$. This distribution is described by a spatially-varying map of normal distributions in our approach. We store and process maps of mean disparity $\mu_i(\mathbf{x})$ and their confidence $\beta_{i,c}\sigma_i^{-2}(\mathbf{x})$ at position $\mathbf{x}$. The factor $\beta_{i,c}$ is a global per-cue i and per-category c weight that gives higher weights to cues that have shown to work better for certain scene categories. Actual values were determined empirically. We now briefly explain the $n_\mathrm{c} = 6$ cues we use. While the input sequence might have an arbitrary spatio-temporal resolution, the typical resolution to store each cue p_i is 300×170 pixels at 3 Hz, which will later be upsampled in space and time by the

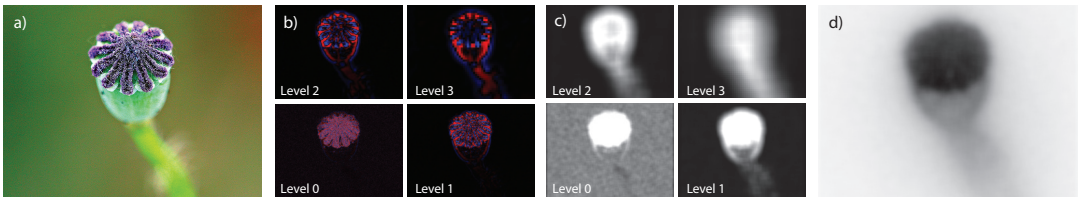


FIGURE 6.5: *Cue extraction for defocus builds a Laplacian pyramid* (b) *of the input image* (a)*. Here, the red/blue color coding represents positive/negative values. The next step is thresholding and blurring of the absolute Laplacian per level* (c)*. The final disparity map* (d) *is created by collapsing the pyramid as described in Sec. 6.4.1.*

pairwise fusion (Sec. 6.5.4). We refer to frames of the image sequence holding depth cues as *keyframes*.

Our cue extraction is conceptually similar to other approaches, but differs with respect to previous work in two ways: First, that all cues can be processed in time linear in the number of pixels and in parallel using common GPU functionality, and second, that they provide an additional measure of per-pixel confidence.

6.4.1 Defocus

Scenes imaged with a finite-size aperture are increasingly blurry at image locations with distances different from the distance of the focal plane. Notably, the defocus only indicates a difference of distance to the focal plane, but not the sign. For the cue to be effective, the image has to contain this depth-of-field, which mostly occurs in images taken with a larger aperture for nearby objects. Depth-from-defocus is computed by measuring the local frequency content around a pixel [Pentland, 1987]. Areas with only low-frequency content are considered out of focus. We use a Laplacian pyramid in multiple passes but constant amortized time per pixel (Fig. 6.5, b). The Laplacian acts as a bandpass while preserving spatial locality and its absolute value can be interpreted as the integral over one octave of the frequency spectrum [Darrell and Wohn, 1988]. On each level of the pyramid, we first soft-threshold the absolute value of the Laplacian up to 0.02 using a sigmoid and then blur the resulting per-level map with a box kernel of size 7×7 (Fig. 6.5, c). The thresholding is required to avoid interpreting high-contrast features (such as edges) as being more in-focus. We then collapse the pyramid by summing the contributions of all levels for each pixel, leveraging hardware-accelerated texture interpolation.

Out-of-focus regions are assumed to lie behind in-focus regions. This assumption, which is not always valid (Fig. 6.14, b) but nevertheless common in the absence of additional information [Lin et al., 2013; Valencia and Rodriguez-Dagnino, 2003], corresponds to images with focused objects in front of a defocused background (Fig. 6.5, a). Consequently, sharp regions map to a disparity of 0 and sufficiently blurred regions to a value of 1 (Fig. 6.5, d).

Confidence for defocus is inversely proportional to disparity. This is motivated by the fact that high-frequency regions can only stem from scene content close to the focal plane, while there is an intrinsic ambiguity for low-frequency regions: either the depicted object is out of focus or it does not contain any high-frequency details (e. g., a plain-colored wall) [Lin et al., 2013]. We found this cue to work better when we additionally reduce the overall confidence if no defocus blur is present in the image. In order to determine if in-focus features dominate the image, we simply calculate the mean disparity of this cue by employing a MIP map.

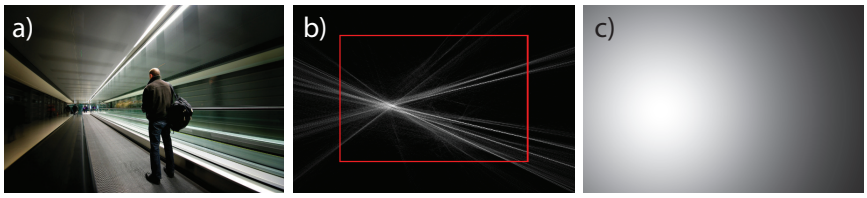


FIGURE 6.6: *Vanishing points are determined by splatting a line primitive* (b) *for each multi-scale edge pixel of the input image* (a)*. The red rectangle indicates the original image boundary. The final disparity map* (c) *is a radial gradient centered at the estimated vanishing point (cf. Sec. 6.4.3).*

6.4.2 Aerial Perspective

Distant objects in images showing a landscape-scale range of depth undergo changes in appearance due to atmospheric scattering. This typically results in a depth-dependent loss of luminance contrast and a color shift towards blue, which can be analyzed to infer depth [Cozman and Krotkov, 1997; Fattal, 2008; Gibson et al., 2013]. As the *Cr* channel of the *YCbCr* color space separates low-frequency from high-frequency wavelengths, we use its inverse as the disparity map of this cue [Tam et al., 2009] in constant time, parallel for all pixels. Pixels with little local contrast in their vicinity (low variance) have higher confidence. Local variance is efficiently estimated using a Laplacian pyramid (cf. Sec. 6.4.1).

6.4.3 Vanishing Points

Perspective projections of parallel 3D lines cross in a 2D vanishing point. If dominant lines are visible in an image, their point of convergence is a strong depth cue we would like to exploit as well. We use an approach based on edge extraction and line accumulation [Barnard, 1983]. First, edge orientation is found at multiple image scales [Ma and Manjunath, 2000] and edge strength is measured by counting the number of scales at which the edge is present. Next, all pixels along a line elongating the orientation of every edge pixel are incremented by splatting a line primitive with additive blending (Fig. 6.6, b). The value of the line increases linearly with the distance to the pixel creating this line. This gradient is required, as vanishing points are more stable if they result in agreement with other lines at an image position far away from the respective pixel causing them. Finally, the pixel in the accumulated line-image that has the highest response to a Harris corner detector is considered the vanishing point pixel. This pixel is found using a parallel reduction.

The drawing area of the accumulated line-image is extended by a factor of 1.5 in both width and height compared to the input frame (red rectangle in Fig. 6.6, b). This way, vanishing points lying a reasonable distance outside the image boundaries can be detected. We found the recovery of vanishing points with positions further away from the image boundaries to become unstable in practice, while additionally only indicating a diminishing depth gradient.

The vanishing point itself is additionally low-pass filtered in time using a temporal cut-off of 0.5 Hz. Disparity is created according to this vanishing point using a radial gradient that is 1 at the vanishing point and 0 at the pixel farthest away from this point (Fig. 6.6, c). Confidence is computed by the curvature of the accumulated value: If all lines concentrate on a single pixel, the confidence is high and the vanishing point is reliable. If multiple vanishing points are found or if the accumulated lines do not concentrate in a small region, the cue is considered less confident. While images

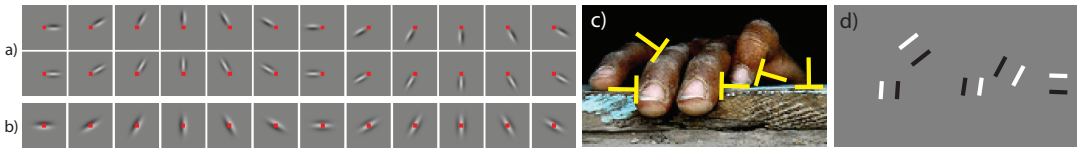


FIGURE 6.7: *Occlusions are found by first convolving the input image with a filter bank* (a, b; *the pixel under consideration is marked red) and then combining the responses to detect T-configurations* (c)*, leading to a sparse map of depth gradients* (d)*. In* a), b) *and* d) *a grey pixel indicates the value zero. The first row in* a) *and the first six images in* b) *show odd kernels for detecting edges, while the bottom row in* a) *and the last six images in* b) *show even filters for detecting lines. Note that the centered kernels in* b) *can be used to produce responses identical to those of the kernels in* a)*. Individual processing steps are explained in Sec. 6.4.4.*

can contain multiple vanishing points, we found it more stable in practice to only pick the dominant one.

6.4.4 Static Occlusions

Occlusion is a strong depth cue that works on all depth scales: If an object A occludes object B, A is closer. However, occlusion is only a relative cue and furthermore cannot be measured directly, only inferred. Occlusions are found by detecting T-junctions of edges and lines. This is done by convolving the image with a bank of separable filter kernels. 24 kernels are necessary to detect incident edges (Fig. 6.7, a, top row) and lines (bottom row) with an angular spacing of 30 degrees at a single scale. Note, that the same response can be created by convolving the image with only 12 centered kernels (Fig. 6.7, b) and then offsetting and/or inverting the resulting responses. We are interested in filter responses at different scales and for this purpose implement filters of increasing size by executing same-sized (15-tap) oriented 1D filters on an image pyramid. The approach of Michaelis and Sommer [1994] is used to detect T-configurations based on these responses. As occlusion only indicates ordering, not absolute disparity, it cannot directly produce disparity and confidence, but produces sparse spatial disparity gradients with high confidence. More precisely, if a T-junction is found (Fig. 6.7, c) at position $\mathbf{x}$ with a vertical bar in direction $\mathbf{d}$ at scale s, a line orthogonal to $\mathbf{d}$ with length $10s$ is drawn with high confidence (we use a constant value of 10 in our implementation) and a positive gradient at $\mathbf{x} + s\mathbf{d}$ and with a negative gradient at $\mathbf{x} - s\mathbf{d}$ (Fig. 6.7, d).

6.4.5 Motion

Several different depth cues are related to motion. Particular observer motions result in typical depth patterns and typical motions in the scene allow predictions about the relative depth of objects. In this work we use the computationally most simple cue that works based on optical flow alone. First, optical flow $\mathbf{f}(\mathbf{x})$ is computed between consecutive frames using a GPU implementation of Lucas-Kanade [Lucas and Kanade, 1981] registration. Although the output of the stereo cues is at low temporal resolution, the flow is computed at the full temporal, but reduced spatial resolution of the input image sequence, as we found flow between consecutive frames to work more reliably than registration of stronger deformations. Flow is augmented by a confidence map $\sigma_{\mathbf{f}}^{-2}(\mathbf{x})$, computed from the local luminance variance of the respective input frame: Flow in featureless regions is considered unreliable. $\mathbf{f}$ is later also used for temporal upsampling and propagation (Sec. 6.5.4).

To determine a disparity and confidence map for each keyframe, the confidence-weighted flow average is removed from the flow, leading to a motion residual

$$\mathbf{f}_r(\mathbf{x}) = \mathbf{f}(\mathbf{x}) - \sum_{\mathbf{x}'} \frac{\sigma_{\mathbf{f}}^{-2}(\mathbf{x}')\mathbf{f}(\mathbf{x}')}{\sigma_{\mathbf{f}}^{-2}(\mathbf{x}')},$$

where the weighted sum over all pixels in the current keyframe is efficiently determined by employing an image pyramid. The residual motion magnitude $\|\mathbf{f}_r(\mathbf{x})\|$ is used as an estimator for motion parallax and finally mapped to disparity, such that fast moving objects are closer. Confidence of this cue is determined by

$$\sigma_{\text{Motion}}^{-2}(\mathbf{x}) = \sigma_{\mathbf{f}}^{-2}(\mathbf{x}) n^{-1} \sum_{\mathbf{x}'} \|\mathbf{f}_r(\mathbf{x}')\|,$$

where n is the number of pixels in the keyframe. Here, the average residual motion magnitude of the keyframe serves as a global indicator that motion parallax is present.

6.4.6 User Input

Optionally, user input can be included as another depth cue to augment traditional manual stereo painting with automatic inference in the propagation. A user simply paints a disparity and confidence map and the system includes this additional cue into the inference. No results in this chapter were produced using any manual intervention, except for Fig. 6.12.

6.5 Cue Fusion

Cue fusion combines evidence from cues over space and time with the scene-specific prior (Fig. 6.8). Here, we will first explain the use of maximum likelihood estimation (MLE) to fuse evidence from multiple cues in a single pixel. Second, we extend the idea to include priors, yielding a maximum a posteriori (MAP) estimate. Next, we describe an iteratively reweighted variant of the estimate to make it robust to outliers and contradicting cues. Finally, we include interactions over time and space, and compute them using efficient edge-aware filtering.

6.5.1 Unary Estimate

The unary estimate predicts the most likely value, given multiple observations with different levels of confidence. For a pixel $\mathbf{x}$, the MLE estimate of disparity $\mu_{\text{MLE}}(\mathbf{x})$ is the confidence-weighted average of disparity means

$$\mu_{\text{MLE}}(\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{i=1}^{n_c} \mu_i(\mathbf{x}) \beta_{i,c} \sigma_i^{-2}(\mathbf{x}),$$

where Z is the normalizing partition function. Furthermore, the MLE of the confidence simply is

$$\sigma_{\text{MLE}}^{-2}(\mathbf{x}) = \sum_{i=1}^{n_c} \beta_{i,c} \sigma_i^{-2}(\mathbf{x}). \tag{6.3}$$

This approach was taken in computer vision for measurements in the presence of sensor uncertainty [Szeliski, 1990] but not for 2D-to-3D conversion.

FIGURE 6.8: *Cue fusion* (left to right). *Here, unary fusion combines confident occlusion, aerial perspective and defocus. The prior overrides values in the sky. Inconclusive evidence between prior and other cues is resolved by iterated re-weighting. The pairwise step propagates confident estimates to other locations, preserving space-time luminance discontinuities and eliminating low-confidence noise.*

6.5.2 Prior

Priors are included in the fusion using Bayesian inference (Sec. 2.1.5), which states that the probability distribution $p(h|e)$ of the hypothesis h given the evidence e is $p(h|e) = p(e|h)p(h)p^{-1}(e)$ [Knill and Richards, 1996]. A prior is included as an additional observation $\{\mu_0, \sigma_0^{-2}\}$, producing the MAP estimate of disparity

$$\mu_{\mathrm{MAP}}(\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\left(\mu_0(\mathbf{x})\beta_{0,c}\sigma_0^{-2}(\mathbf{x}) + \sum_{i=1}^{n_c}\mu_i(\mathbf{x})\beta_{i,c}\sigma_i^{-2}(\mathbf{x})\right).$$

The MAP estimate of variance $\sigma_{\mathrm{MAP}}^{-2}(\mathbf{x})$ is computed by extending the sum of the MLE confidence (Eq. 6.3):

$$\sigma_{\mathrm{MAP}}^{-2}(\mathbf{x}) = \beta_{0,c}\sigma_0^{-2}(\mathbf{x}) + \sum_{i=1}^{n_c}\beta_{i,c}\sigma_i^{-2}(\mathbf{x}).$$

In practice, the prior extracted in the pre-process (Sec. 6.3) that expresses information for all possible appearances at a location (conditioned prior $\{\bar{\mu}_{0,i}, \bar{\sigma}_{0,i}^{-2}\}$; Eq. 6.1, Eq. 6.2) is used for an image with a specific appearance at a specific location (unconditioned prior $\{\mu_0, \sigma_0^{-2}\}$). Let $L(\mathbf{x}) \in \mathbb{R}^2 \to \mathbb{R}^3$ be this appearance, a simple RGB image. We denote mean and variance of the final unconditioned priors as $\mu_0(\mathbf{x}) = \mathtt{fetch}(\bar{\mu}_0, (\mathbf{x}|L(\mathbf{x}))$ and $\sigma_0^{-2}(\mathbf{x}) = \mathtt{fetch}(\bar{\sigma}_0^{-2}, (\mathbf{x}|L(\mathbf{x}))$. The function $\mathtt{fetch}(X, \mathbf{y}) \in \mathbb{R}^5 \to \mathbb{R}$ is the 5D linear filtering of a grid X at position $\mathbf{y}$. For efficiency, we store the prior as a 2D array (spatial domain) of 3D textures (appearance domain). As linear filtering is separable, this texture is read using four 3D linearly-filtered hardware-accelerated interpolations in the appearance domain followed by spatial interpolation.

6.5.3 Robust Estimate

If multiple high-confidence cues (including the prior) indicate different disparities, not all can be correct and at least one of them has to be considered an outlier. As MLE and MAP estimates for Gaussian noise models are generalized least-squares fits, they do not perform well in such conditions [Green, 1984], as a single outlier quadratically skews the entire solution. Consider an example of two cues (e. g., focus and aerial perspective) and the prior that indicate a blurry blue pixel in the top to be far away, and a single cue (e. g., motion) to indicate it is close, all with the same confidence. A least squares-fit would indicate a medium disparity value. A more robust fit would result in a distant disparity and ignore the other cue as an outlier. This can be achieved by an iteratively reweighted MAP estimation. In each step (3 in our implementation) a weighted MAP is computed. In the first iteration, the weight is 1 for all evidence. In later iterations, the weight of evidence not supporting the MAP estimate of the previous iteration is decreased. Evidence does not support the

estimate, if it is very different from it. The Cauchy weight function [Green, 1984] is used to control the reweighting.

6.5.4 Pairwise Estimate

The disparity at one space-time location $\mathbf{x}$ also depends on evidence from other pixels at nearby space-time positions $\mathbf{y}$. This serves both as an additional regularization constraint and as an opportunity to share information between less confident and more confident space-time locations. We model these dependencies using a variant of a fully-connected conditional random field. The pairwise term contains a *domain* weight (disparity of nearby pixels should be similar) and a *range* weight (pixels with similar luminance values should have similar disparity),

$$v(\mathbf{x},\mathbf{y}) = \mathcal{N}(\|\mathbf{x}-\mathbf{y}\|_{\mathbf{f}},\sigma_\mathrm{d})\mathcal{N}(I(\mathbf{x})-I(\mathbf{y}),\sigma_\mathrm{r}),$$

where I is the monocular image intensity [Tomasi and Manduchi, 1998; Kopf et al., 2007; Richardt et al., 2012]. Here, we assume the images have been motion-compensated, i. e., $\|\mathbf{x}-\mathbf{y}\|_{\mathbf{f}}$ is the spatial distance of $\mathbf{x}$ and $\mathbf{y}$ moved to the time coordinate of $\mathbf{x}$ along $\mathbf{f}$. We set this distance to infinity if they are not related by optical flow. Then the final inference that combines spatially-varying cues and priors with confidence maps and interactions of pixels in space and time is

$$\mu(\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\int_\Omega v(\mathbf{x},\mathbf{y})\sigma_\mathrm{MAP}^{-2}(\mathbf{y})\mu_\mathrm{MAP}(\mathbf{y})\mathrm{d}\mathbf{y} \tag{6.4}$$

with confidence

$$\sigma^{-2}(\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\int_\Omega v(\mathbf{x},\mathbf{y})\sigma_\mathrm{MAP}^{-4}(\mathbf{y})\mathrm{d}\mathbf{y}, \tag{6.5}$$

where Ω is the entire space-time domain. This inference is realized in three steps: i) Pixel-wise pre-multiplication of the mean disparity map μ_MAP by its confidence map σ_MAP^{-2}; ii) edge-aware blurring of both the pre-multiplied mean disparity and confidence maps in time and space; iii) per-pixel division of the propagated mean disparity by its confidence [Knutsson and Westin, 1993].

Steps i) and iii) are trivially parallel and equivalent to compositing using pre-multiplied alpha. For propagation in time, the two nearby keyframes are first motion-compensated and then blended [Miksik et al., 2013]. Recall that we compute the flow in full temporal resolution in the depth-from-motion cue component. For motion compensation, we forward-concatenate the flow from the past keyframe and backward-concatenate the flow from the future keyframe and use this flow to warp depth from the respective keyframes into the current frame. Warping disocclusions are filled using push-pull from a Gaussian MIP map. The backward flow is approximated using the negated forward flow, assuming motion is linear on small time scales. The result is then linearly blended using the temporal distance to the future and past keyframe as weights. The output of this step is at full temporal, but still at low spatial resolution. For propagation in space, a two-channel bilateral grid [Chen et al., 2007] with 8 layers and the full spatial resolution is used. Confidence-weighted disparity and confidence values are inserted into the layers of that grid using the final image intensity I as a guide with a standard deviation of $\sigma_\mathrm{r} = 0.1$. This grid is then blurred using a standard deviation of $\sigma_\mathrm{d} = 0.5\,\mathrm{deg}$ using a Gaussian MIP map. Next, the bilateral grid is upsampled to the desired high resolution, using the

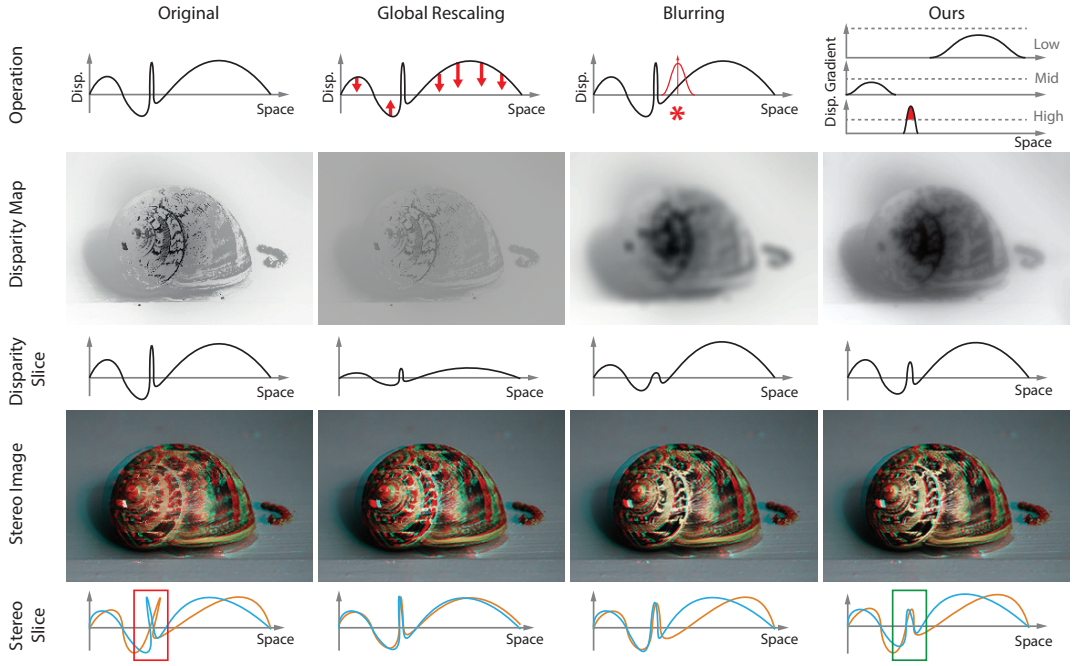


FIGURE 6.9: *Disparity maps and stereo images without* (first column) *and with different approaches* (second to fourth column) *to enforce the disparity gradient limit.* First column: *The original disparity map contains gradients exceeding the limits, resulting in fold-overs in the stereo signal* (red box). Second column: *A global linear rescaling resolves the issue, but results in a loss of overall depth contrast.* Third column: *Simple low-pass filtering is a natural way to reduce exceeding gradients, but comes at the cost of the loss of fine details.* Fourth column: *Our approach prevents fold-overs while at the same time retaining the global depth range and fine-scale details not exceeding the disparity gradient limit* (green box).

high-resolution luminance as a guide. After this step, the filtered, high-resolution disparity-confidence product is finally divided by the filtered confidence component.

6.6 Stereo Image Generation

The final step converts the acquired disparity maps into a stereo image pair. This step is a standard 2D-to-3D procedure for which many alternatives exist. We use grid-based image deformation [Mark et al., 1997] with a cell size of one pixel.

Before converting, however, we assure that the upper disparity gradient limit is maintained. Our disparity is produced by an automatic process and contains disparity with high spatial frequencies (Fig. 6.9, left column) that is important for the vivid and natural appearance. Consequently, the result may contain areas which are too distorted to be fused or even overlap (red box in Fig. 6.9). In particular, the result may contain fold-overs, where space runs backwards to create an overlap. We correct for this issue in a post-process as follows. First, a Laplacian pyramid of disparity [Didyk et al., 2012] is produced, which contains gradients of disparity at multiple scales (Fig. 6.9, top right). Gradient values outside the level-dependent fusible disparity range [Howard and Rogers, 2012; Kane et al., 2014] (dotted lines in Fig. 6.9, top right) are clamped (red area in Fig. 6.9, top right). Finally, the resulting pyramid is collapsed into a new disparity map (Fig. 6.9, center right) that is fusible (green box in Fig. 6.9, lower right). Note, how the above is not equivalent to global rescaling, nor is it equivalent to blurring. Both are options to fit stereo content into the gradient limit range, but would result in a reduced overall depth impression or in

loss of fine details (Fig. 6.9, second and third column). Instead, our processing only removes disparity variations that are too strong for their spatial extent.

6.7 Evaluation

Example results of our real-time system are shown in Fig. 6.10. All are produced at 35 fps on a Geforce GTX 780 with an Intel Xeon E5-1620 CPU. A timing breakdown can be found in Tbl. 6.1. Results for video are seen in Fig. 6.11. An example comparison between our cue-guided manual 2D-to-3D conversion and a conventional scribble interface is seen in Fig. 6.12. We found that our system works well over a range of scenes, while other approaches are more specific to a certain class, e. g., static street-level outdoor images. While other approaches are specialized to a specific cue (like vanishing points), certain motion (like rigid), a certain shape (like ground plane), or requiring that the image is similar to an image in a database, our technique relies on a greater variety of pictorial depth cues combined with priors based on scene types. Finding a balance between prior information and individual cues is an important component of our system (Fig. 6.13, a-d). To use a prior, the scene needs to be classified, and if classification fails, disparity quality degrades as seen in Fig. 6.13, e-h. Failure cases are discussed in Fig. 6.14.

TABLE 6.1: *Computation time for a keyframe (every ca.* 3 Hz*) and for every non-keyframe (more than* 30 Hz*) at a resolution of* 1280 × 720*. Time for the actual computation granularity used is shown in bold.*

Part	Step	Time		Res.
		10 f.	1 f.	
Cue	Aerial per.	**2 ms**	0.2 ms	300 × 170
	Defocus	**8 ms**	0.8 ms	300 × 170
	Van. points	**18 ms**	1.8 ms	300 × 170
	Motion	**9 ms**	0.9 ms	300 × 170
	Occlusions	**11 ms**	1.1 ms	300 × 170
Opt. flow		58 ms	**5.8 ms**	300 × 170
Fusion	Robust MAP	**13 ms**	1.3 ms	300 × 170
	Temp. prop.	30 ms	**3.0 ms**	300 × 170
	Spatial prop.	46 ms	**4.6 ms**	1280 × 720
Warping		80 ms	**8.0 ms**	1280 × 720
		275 ms	27.5 ms	

6.7.1 Cue Influence Analysis

In order to gain insights into the influence of the cues and the prior on the resulting stereoscopic conversion we analyzed the results of our system for 83 videos with 80 keyframes each. Fig. 6.15, a lists the mean confidence of each cue and the prior. One can observe that the contribution of the prior on the result is about 50 %, while the cues contribute the other half of the depth information. Fig. 6.15, b gives each cue's tendency to be an outlier by showing the confidence-weighted deviation of its disparity estimate from the robust unary and the final pairwise estimate. One can observe that the deviation is fairly uniform across the cues, while the pairwise propagation step of our system tends to increase the deviation in order to perform the space-time regularization. Finally, Fig. 6.15, c shows the normalized confidence distribution of each cue over the disparity range. We can observe that the occlusion cue is mostly covering near distances, while the defocus, vanishing point and motion

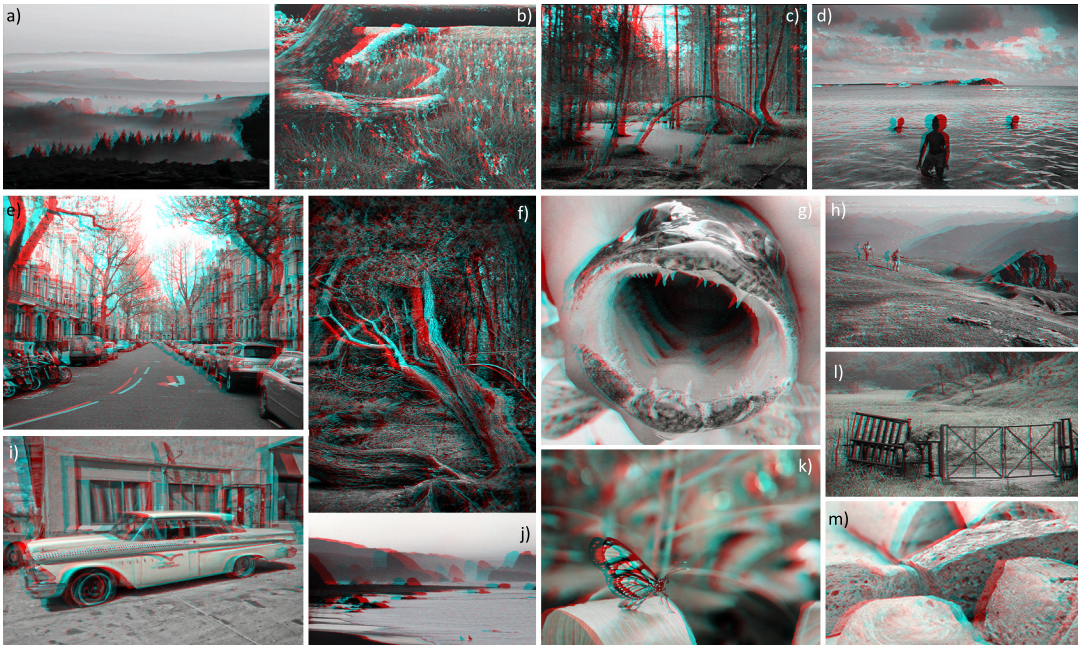


FIGURE 6.10: *Results for static images. (a) (Aerial perspective, occlusion) A typically good result as luminance edges give a good indication for depth edges. (b) (Prior) The fine details detach flowers from the ground. The overall ground plane is perceivably non-linear in depth, a typical artefact of our approach. (c) (Prior) The bright sky, the dark trees, the ground plane in the foreground and the forest-typical color-disparity relation in this image allow a plausible, detailed result. (d) (Prior) Classification into coast is easy by the colors. Human shapes are distinct from their context due to the edge-aware pairwise propagation. (e) (Vanishing point, prior) This is a typical street-level scene well covered by other approaches. Our result reproduces the side walls, the sky and the ground plane, but also includes fine details. (f) The twigs indicate occlusions, otherwise this image is dominated by the prior. Disparity is considerably wrong, but the fractal distribution of disparity combined with a correct tendency from the prior produces a consistent stereo look. (g) This image works, because the disparity contrast at the strongest depth discontinuity is correct due to a generic vertical-gradient prior. (h) (Aerial perspective, prior). The color-dependency of the prior correctly places depth edges on the hill's horizon lines at all distances. (i) (Occlusion, vanishing point, prior) It can be noted that the ground plane in the front is not a plane in disparity. (j) (Aerial perspective, prior) (k) An image following no prior, where defocus is found as the relevant cue, detaching the butterfly from the backdrop. (m) (Occlusion) (l) An open country prior is fused with occlusion from the fence preserving fine details with good depth discrimination.*

FIGURE 6.11: *Our 2D-to-3D video result with motion-compensated filtering provides temporally stable stereo with fine details.*

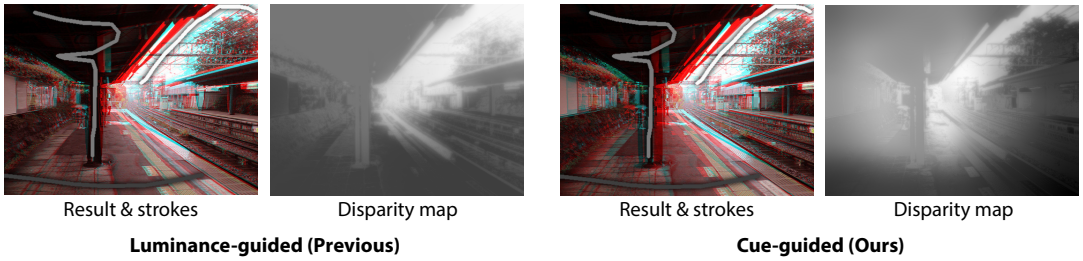


FIGURE 6.12: *Manual 2D-to-3D stereo conversion without* (left) *and with* (right) *using our cue fusion. Our approach results in a better disparity layout and keeps details, such as the wires.*

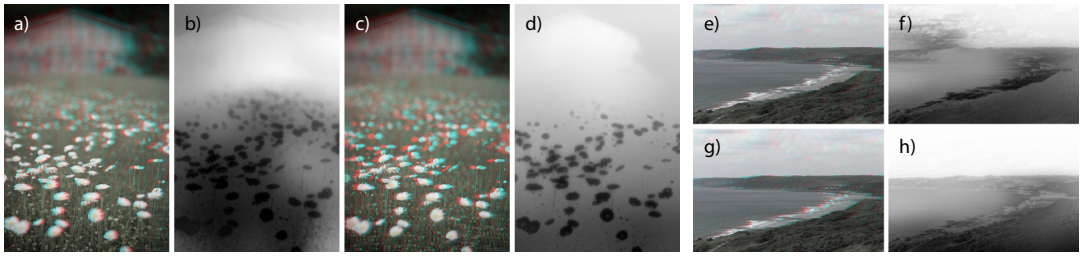


FIGURE 6.13: *Result* (a) *and depth map produced by cue fusion without priors* (b)*, and including the prior for open country* (c and d)*. The defocus cue has identified the sharpness gradient complemented by the prior. An image* (e) *was classified to show mountains, resulting in a disparity map* (f) *that is more vertical as seen from the low vertical contrast and the light-grey beach is mapped to a near depth. With correct classification as coast* (g)*, the beach will be placed at medium depth* (h)*.*

FIGURE 6.14: a) *High-contrast textures can cause problems in the cue extraction as well as the cue fusion phase. Here, the occlusion module detected several T-junctions in the butterfly wing and hallucinated depth gradients. This misinterpretation cannot be compensated by the pairwise fusion, since it does not distribute the available depth information across the whole object, but rather stops at the luminance edges. This leads to false-positive depth edges in the disparity map.* b) *If an assumption made in a cue extraction module is violated, the module may produce wrong disparity values with high confidence. If only a small number of cues is present in the input video, there is not much reliable information to compensate for that. In this case, the assumption of the defocus cue, that blurred regions are distant, is violated. Since there is no other strong cue present, this leads to a large disparity in the foreground.* c) *The motion cue fails, because the walking subjects cover the image in large part. This leads to residual motion, whose magnitude is low for the subjects and high for the background, hence turning the latter into foreground.* d) *For a camera rotating around an object, both close and far points with high velocity get classified as close.*

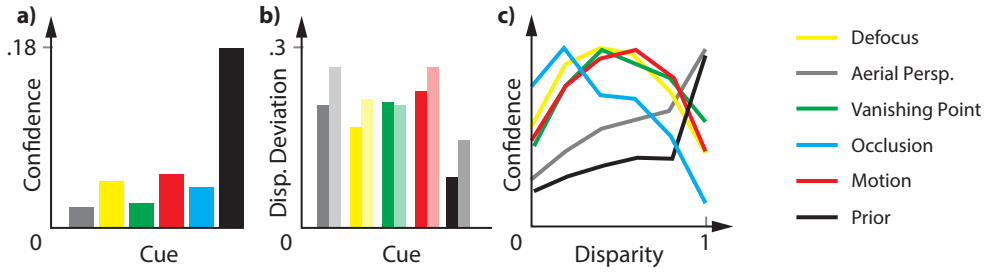


FIGURE 6.15: *Cue influence analysis:* a) *Mean confidence.* b) *Mean deviation of each cue's disparity estimate from the robust unary* (dark) *and the final pairwise estimate* (light)*. Occlusion is not listed here, as it only provides disparity gradients.* c) *Normalized confidence distribution over disparity.*

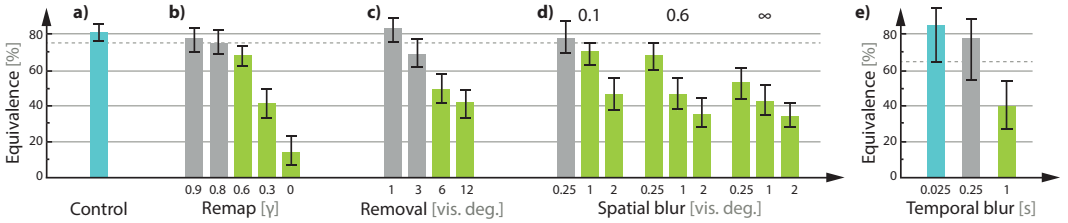


FIGURE 6.16: *Perceptual experiment analysis (Sec. 6.7.2): The horizontal axis shows different bars for different distortions. The vertical axis is equivalence in percentage. A high value means that the distortion is more equivalent to a reference. A green bar has a significantly different equivalence compared to how equivalent the reference is to itself, which is only ca. 80%, not 100%. Bars are grouped by distortions. Inside each group the distortion is the same, just more or less strong in one (*b, c *and* e*) or two (*d*) respects.*

cues have their strongest influence in the mid-range. The aerial perspective cue as well as the prior mostly cover the larger distances. We conclude that our cues provide a balanced mixture of sources of information. In our versatile test dataset all cues provide important information and tend to complement each other, while our data-driven prior gives strong indications whenever there is not enough evidence from the cues alone.

6.7.2 Validating Plausible Disparity

We would like to know to what extent the three properties of perceptually plausible disparity, which motivate our approach (Sec. 6.1), are applicable to complex images. To this end, we run perceptual experiments, in which we intentionally reduce physical disparity in these aspects [Kellnhofer et al., 2015].

Experiment

Participants were asked if they consider a physical and a distorted disparity stimulus visually equivalent or not. The physical disparity in our stimuli is distorted by one out of four simple operations: i) remapping by a power curve with a gamma value of $r_1 \in \{0.9, 0.8, 0.6, 0.3, 0\}$, ii) entire removal of a disparity from circles of radius $r_2 \in \{1, 3, 6, 12\}$ visual degrees followed by luminance-based edge-ware inpainting that restores structure but not disparity values, iii) edge-aware spatial blurring with a spatial std. dev. of $r_{3,1} \in \{0.25, 1, 2\}$ visual degrees and range Gaussian std. dev. of $r_{3,2} \in \{0.1, 0.6, \infty\}$ in the intensity range from 0 to 1, as well as iv) temporal blurring with a std. dev. of $r_4 \in \{0.025, 0.25, 1\}$ seconds. The original image or movie in comparison to the reference is used as a control group. 17 participants took part in the experiment, which comprised of 2 repetitions for each of the 4 videos or images being presented with 1 placebo, 5 different remappings, 4 removals, 3×3 spatial blurs and 3 temporal blurs yielding the total of $2 \times 4 \times (1 + 5 + 4 + 3 \times 3 + 3) = 64$ trials. In each trial, participants were shown the reference image and a distorted variant in a randomly shuffled vertical arrangement for 3 seconds and were asked if they provide an equivalent stereo impression or not.

Results and Discussion

We compute sample means and confidence intervals (binomial test, 95% CIs, Clopper-Pearson) for the percentage of trials in which a distorted and an original are considered equivalent (Fig. 6.16). The control group that is not distorted at all (placebo),

is considered equivalent to the reference in 79.0% $\pm$ 4.0% of the cases (Fig. 6.16, a). Consequently, a reduction that is equivalent will result in a measure of equivalence of ca. 80% in the best case, not 100%. Equivalence is rejected using a two-sample t-test (all $p < 0.01$). Additionally, the effect of reduction can be seen from comparing their CIs to the control group, in particular, its lower bound (Fig. 6.16, dotted line).

Remapping values for $r_1 \leq 0.6$ (stronger deviation from identity) are significantly nonequivalent (Fig. 6.16, b), indicating (but not proving) that more subtle remappings might be equivalent. Our approach does only reproduce disparity up to such a smooth remapping. Not reproducing objects as large as $r_2 = 6$ vis. deg. or larger are significantly nonequivalent (Fig. 6.16, c), indicating that removal of smaller objects might not be objectionable. In our approach, some objects do not get resolved because neither a cue nor a prior provides evidence for its depth. As long as such objects are consistently embedded into the environment, which typically happens due to our luminance-based edge-aware upsampling, the proper values of depth are not mandatory.

For blurring (Fig. 6.16, d), not respecting edges ($r_{3,2} = \infty$), or edge-stopping blurring ($r_{3,2} = 0.6$ and $r_{3,2} = 0.1$) with a spatial Gaussian of std. dev. $r_{3,1} \geq 1$, resp. $r_{3,1} \geq 2$ vis. deg. is not equivalent. This indicates that the slightly larger spatial extent and similar range support used in our approach produces a functionally equivalent result.

For temporal blurring (Fig. 6.16, e) all reductions with a temporal Gaussian of std. dev. $r_4 \geq 1$s have been found visually non-equivalent. This indicates that temporal disparity sampling can be surprisingly sparse if it is motion-compensated, as in our approach, where disparity is computed only for keyframes at ca. 3 Hz which is likely faster than the value required for equivalence. This outcome indicates that in natural images, even more edge-aware spatial blurring and temporal filtering is tolerated than what was reported for disparity-only stimuli by Kane et al. [2014]. While the reductions in our experiment (and application) might introduce conflicts between disparity and pictorial cues, the latter seem to play the dominant role in depth perception, and tolerance for disparity reduction is higher. Edges at larger depth discontinuities must be preserved (Fig. 6.16, d), and in the temporal domain (Fig. 6.16, e) disparity should follow the image flow, while the temporal update of specific disparity values can be sparse.

6.7.3 Perceptual Comparison Study

We would like to know if the results produced in real time by our method are preferred over other approaches. Therefore, image pairs produced by our method and one of three previous methods were presented using Nvidia 3D Vision active shutter glasses on a 27″ Asus VG278HE display with a resolution of 1920×1080 pixels at a viewing distance of 60 cm under normal office lighting. 10 participants (all male, 23 to 30 years old) took part in the study. All of them had normal or corrected-to-normal vision and passed a stereo-blindness test. The subjects were naïve to the purpose of the experiment. Overall, 77 image pairs were used. Each pair was presented as a random horizontal arrangement and participants were asked which image provides a better 3D impression. The images have been produced using methods proposed by Saxena et al. [2009], Cheng et al. [2010], and Karsch et al. [2014]. In our study, we include results on our images for the method of Cheng et al., images and depths provided by the original publication for the method of Saxena et al. and a mixture of both for the method of Karsh et al.

To produce results for our images the method of Karsh et al. was trained using 400 outdoor images from the Make3D dataset [Saxena et al., 2009] as done in the original paper. Our main goal in this study was to maximize the participants' performance in seeing differences between the methods. Therefore we chose to use static images instead of videos, since human disparity sensitivity decreases with motion [Kane et al., 2014; Kellnhofer et al., 2016] and participants were less likely to overlook artifacts. Our method is preferred over the method of Cheng et al. in 69.6 % $\pm$ 3.3 % (0.95 confidence intervals, binomial) of the cases, over the one of Saxena et al. in 64.4 % $\pm$ 7.0 % and over the approach of Karsch et al. in 54.5 % $\pm$ 3.6 % of the cases. All comparisons are statistically significant ($p < 0.02$). Comparing our result and the method of Karsch et al. on a subset containing their images leads to a significant preference for their results (31.3 % $\pm$ 8.1 % prefer ours) while comparing on a subset only containing our images provides significant preference for our method (60.0 % $\pm$ 3.9 %). This can be attributed to a non-optimal training set for certain images used in the study.

We conclude that we can outperform real-time and offline 2D-to-3D conversion methods for general imagery, while the performance of data-driven offline methods highly depends on the training data used.

6.7.4 Quantitative Evaluation

The final quality of a stereo image arises from the complex interactions of monocular and binocular stereo cues, for which no computational model is available. The perceived error of a 2D-to-3D stereo conversion consequently correlates only very little with the predictions of classic image quality metrics such as the peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) index when they are applied to the disparity maps [Kellnhofer et al., 2015]. Merkle et al. [2009] show that more meaningful quality predictions can be obtained when the reconstructed disparity is actually applied to generate stereo-image pairs and those are compared to the ground-truth images. Tbl. 6.2 nonetheless lists the numerical error with respect to the ground truth NYU (Kinect sensor; well-aligned key luminance and depth edges) and Make3D (laser scanning; low-resolution depth maps) data sets for the approaches of Cheng et al. [2010], Karsch et al. [2014] as well as ours and a baseline that uses low-frequency fractal noise as a disparity map. We see that according to the PSNR (which is poor in detecting localized disparity distortions and rather assumes their spatially uniform distribution), the approach of Karsch performs best and that most approaches perform better than random, but not on all datasets and according to all metrics. Overall, in terms of SSIM, the margin starts to get smaller. Finally, when using the most recommended metric by Merkle et al. , the differences between all three methods are marginalized.

We conclude that we can achieve similar quality in terms of error numbers as the competitors that either take much longer to compute and/or have a lower user preference. Interestingly, although the visual quality of the fractal baseline stereo-image pairs is clearly not acceptable, the metric predictions (Tbl. 6.2) do not show them as clear outliers in all cases. The fact that, on one hand, we do not intend to reproduce ground truth depth but rather perceptually plausible disparity, and, on the other hand, the given quantitative evaluation clearly does not reflect stereoscopic conversion quality, indicates that our perceptual comparison study (Sec. 6.7.3) provides the most meaningful evaluation results.

TABLE 6.2: *Numerical comparison (larger is better).*

	NYU Range				Make3D			
	Disparity		Image pair		Disparity		Image pair	
Method	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Cheng et al.	9.96	0.72	21.84	0.80	10.30	0.56	16.91	0.42
Karsch et al.	10.77	0.76	21.77	0.80	11.60	0.77	18.40	0.49
Ours	10.18	0.74	21.03	0.78	10.03	0.66	17.02	0.43
Baseline	10.11	0.75	18.20	0.68	8.69	0.69	16.29	0.37

6.8 Conclusion

In this chapter we have proposed a system to infer perceptually plausible binocular disparity from a monocular video stream in real time. Several monocular cues estimate disparity and confidence maps of low spatial and temporal resolution, which are complemented by spatially-varying, class-specific disparity priors. Robust MAP fusion produces stereo image streams with high spatial and temporal resolution. Perceptual experiments favorably compared our approach to existing techniques. Our method reconstructs perceptually plausible disparity and not physical depth, which was inspired from how humans proceed when manually annotating disparity in 2D-to-3D conversion.

Chapter 7

Conclusion

This chapter concludes the thesis. First, we summarize and discuss our contributions in Sec. 7.1. Then we elaborate on possible algorithmic combinations in Sec. 7.2, before developing directions for future work in Sec. 7.3. Closing remarks are given in Sec. 7.4.

7.1 Summary and Discussion

This thesis has demonstrated how artificial intelligence can speed up image-based rendering algorithms and extend their scope. The common methodology in each chapter was to utilize a problem-specific form of intelligence in a pre-process that the actual execution could greatly benefit from. Deep Point Correlation Design (Chapter 3) utilized modern back-propagation to train a generative model. Minimal Warping (Chapter 4) planned the traversal of images based on an efficient approximation of displacements in the space of distribution coordinates. Laplacian Kernel Splatting (Chapter 5) performed a sparsity-enforcing optimization in the space of point-spread functions. Stereo Cue Fusion (Chapter 6) learned a disparity prior with a scene classifier which would guide an efficient probabilistic inference procedure. In each of these cases the application of AI payed off, either in terms of efficiency, result quality, versatility, or their combination. The employed AI technologies differ notably in their degree of sophistication. While uninformed graph traversals and stochastic optimization are rather simple procedures, probabilistic graphical models and deep learning mark the state of the art in AI research.

An important distinction needs to be drawn between Minimal Warping and Laplacian Kernel Splatting on the one hand, and Stereo Cue Fusion on the other hand. Besides plain images, the former techniques require additional per-pixel information, like e. g., depth or motion vectors. The latter method works on images alone, extending its scope significantly. This difference in required input is directly reflected in the respective algorithmic choices.

The following paragraphs discuss the solutions of the individual tasks in image-based rendering that this thesis developed.

Point Pattern Design The deep-learning-based algorithm developed in Chapter 3 optimizes for point pattern creation methods. Our approach employs a generative model which projects random inputs to the manifold of point patterns with prescribed properties. Interestingly, the model is trained without ever sampling the data distribution, i. e., without considering concrete realizations of point patterns. Rather, an indirect, statistical description of the point patterns is provided at train time. This form of *inexact supervision* [Zhou, 2017] allows our system to produce point patterns with characteristics that have been impossible to realize before.

While we have demonstrated how our approach can fulfill many common tasks accurately, we share a limitation with many data-driven approaches: The lack of theoretical guarantees. Simply put, we state the design agenda and hope for modern optimizers to find a good solution. However, most mathematical derivations in the field of sample pattern design also do not provide rigorous evidence, such as we are unaware of a proof that Lloyd relaxation converges in high dimensions. Besides the use cases as an *exploratory* device, in this light, our method can be a valuable tool for the *analysis* of point patterns, eventually moving forward their theoretical understanding. This is due to its property of being a unified and universal framework capable of producing or at least closely approximating any point correlation that can be expressed in terms of an admissible agenda.

Distribution Effects The algorithms developed in Chapter 4 and Chapter 5 are two very different approaches to attack the problem of image-based distribution effect rendering. While Minimal Warping interprets a distribution effect as the sum of many "pinhole" views, i. e., images that do not contain the desired light field integrals, Laplacian Kernel Splatting adopts the view of impulse responses in the form of PSFs. Both interpretations are equivalent, but result in completely different algorithms and vary accordingly in terms of efficiency, quality, and use cases.

A connecting element of both developed algorithms is their generality: Unlike many previous works our methods are not confined to a single effect, but support a wide range of light field interactions, like e. g., the important combination of depth of field and motion blur. Consequently, both methods have to tackle the inevitable curse of dimensionality, resulting in an exponential explosion of computation when effects are combined. Minimal Warping solves this problem by relying on a sparse distribution flow representation. Only a small number of per-pixel pilot samples is required to fill the entire space of image deformations. The view sample tree, however, needs to fill this space densely to ensure that its edges correspond to minimal warps. This is considered acceptable, as the warps themselves are computationally extremely lightweight. Laplacian Kernel Splatting has to deal with a similar problem: The space of PSFs needs to be covered exhaustively as well. By observing that pre-filtering in this space is mostly noticeable for small PSFs, a custom nested sub-sampling structure prevents a combinatorial explosion. Crucially, PSF sampling and sparsification happen in a pre-process, so that the curse of dimensionality mostly affects pre-compute time.

In terms of quality, Minimal Warping trumps Laplacian Kernel Splatting. This is, as Minimal Warping has the core advantage of accurately resolving primary visibility. Each node in the view sample tree is a valid image by itself, as fine-grained occlusions are handled explicitly in the warping procedure. Furthermore, this method allows for a carefully designed layered image representation: Pixels undergo a custom sorting procedure to maximize result quality. In contrast, Laplacian Kernel Splatting – in its synthesis mode – relies on the pre-integration of radiance and opacity per layer. This requires global depth layers as an (intermediate) scene representation, which can adapt to the image content only to a very limited extent. Even though global depth layers are a common approximation both in synthesis [Lee et al., 2009] and reconstruction [Vaidyanathan et al., 2015], their use in the context of distribution effects does inevitably pose problems in particular for semi-transparencies. While carefully extracted disocclusion maps [Lee et al., 2008] can alleviate the artifacts, they come along with rather high computational costs. We can therefore conclude that Minimal Warping successfully masters the task of level-1 visual perspective-taking, while Laplacian Kernel Splatting resorts to an approximation thereof.

In terms of execution speed, Laplacian Kernel Splatting is unsurprisingly ranked first. This is particularly striking when comparing the timing for same-sized combinations of distribution effects. Here, Minimal Warping is up to three orders of magnitude slower than Laplacian Kernel Splatting. However, while this thesis put an emphasis on large-scale distribution effects, the efficiency of Minimal Warping can also be exploited in real-time applications. An example is latency reduction in virtual reality, where warping is commonly used for temporal upsampling [Van Waveren, 2016]. Performing small incremental warps is a perfect match for the high frame rates prevalent in this setting.

Besides the broader range of output sampling modalities that Minimal Warping can handle in comparison to Laplacian Kernel Splatting (e. g., temporal upsampling, stereo, light fields), the former does also comprise a larger set of supported distribution effects. In particular soft shadows and spectral caustics are only discussed in the context of Minimal Warping. Incorporating these effects in the framework of Laplacian Kernel Splatting would require to express them in terms of PSFs. Considering the physics of light transport, this is certainly realizable. However, both soft shadows and in particular spectral caustics have many more degrees of freedom than the other distribution effects discussed in this thesis. Corresponding pre-computed PSFs would span a high-dimensional space resulting from the large variety of possible physical configurations, e. g., the position and shape of light sources, occluders, receivers, and dispersive interfaces. Expressing these variables in terms of deformation flow fields as done in Minimal Warping is vastly more convenient and economic.

2D-to-3D Conversion The method described in Chapter 6 performs a perceptually-motivated conversion from monoscopic to stereoscopic video content. It is based on the observation that plausible stereo is only loosely connected to physical depth. The comparably small-scale image deformations resulting from disparity maps of real-world imagery do not pose a major challenge in the sense of level-1 visual perspective-taking. However, in contrast to the methods described in the previous paragraph, here we do not have access to a layered scene representation. Half-occlusions [Harris and Wilcox, 2009], albeit small-scale, can therefore not pe properly resolved. Our grid-based warping simply stretches the background to fill in missing information. While this is noticeable under close inspection of the individual views, the stereo experience does not suffer.

We found our algorithm to produce images that might have physically incorrect depth, yet, they almost always provide a 3D look due to the agreement to high-frequency luminance features and overall plausible layout. The quality of results our approach produces seems to be rather insensitive to the variety of scenes we explored.

The disparity database used to train our priors was created by painting, as is done in manual stereo conversion. Depending on the problem at hand, working with sensor data instead can be more or less efficient than our pragmatic approach.

In the light of recent ground-breaking developments in 3D scene reconstruction from single images based on supervised deep learning, the question arises if our system is already superseded. To some extent this is certainly the case. When you have the choice between an accurate and a plausible result, you choose accuracy. Current deep architectures tend to come close and even reach real-time execution speed. Combinations with probabilistic models are common [Roy and Todorovic, 2016] and can be accelerated using filtering algorithms similar to ours [Barron and Poole, 2016]. However, the distribution of natural videos is extremely heavy-tailed, i. e., it is very difficult to capture all relevant intricacies using a finite training set.

To solve this problem, strong priors can be beneficial. The depth cue modules of our system essentially encode perceptually relevant physical laws, which can be considered the ultimate prior for an image analysis task in this universe. We therefore conceive that our approach offers a complementary toolkit modern machine learning can benefit from. Hard-coded and hand-crafted parts in a deep learning architecture are admissible as long as it is possible to back-propagate through them. The relevant modules of the system presented in Chapter 6 fulfill this requirement.

All image-based rendering methods developed in this thesis make use of a common approximation: The re-use of shading. Neglecting the *view-dependent appearance* of many materials, shaded pixel information is bluntly recycled in novel views. This results in e. g., highlights and mirrored images stuck to the surfaces of scene objects. When interpreted strictly, this is a failure of level-2 visual perspective-taking. However, perceptual research shows that even humans are not able to reliably reason on specular light-surface interactions [Fleming et al., 2003; Ramanarayanan et al., 2007; Marlow et al., 2012]. While it is possible to take specular shading into account in image-based rendering, it usually requires a clean separation of diffuse and specular components [Lochmann et al., 2014]. A notable exception is the work of Kopf et al. [2013] who apply image deformations in the gradient domain, which tends to de-correlate diffuse and specular shading. We consider these works orthogonal to our warping-based approaches, with one caveat however: Specular shading of surfaces with high curvature result in potentially unbounded deformation flow. In such cases a specular minimal warping algorithm would have to resort to a fall-back strategy, such as plain forward warping.

A separation or de-correlation of diffuse and specular shading does not appear promising for a PSF representation as is adopted in Laplacian Kernel Splatting. In these cases a parametric appearance model would be beneficial. In particular, low-dimensional image-based appearance manifolds [Lischinski and Rappoport, 1998; Maximov et al., 2018] are required to avoid the curse of dimensionality during pre-processing.

7.2 Algorithmic Combinations

A straightforward and intended amalgamation is the use of Deep Point Correlation Design for the sampling tasks of other methods. In Minimal Warping we require sampling in the domains of distribution flow on the one hand and views on the other hand. For the former case any design agenda could be used that leads to a good stratification. The core principle of minimal warps, however, crucially depends on view samples that have very similar distances to their respective direct neighbors. A trivial solution to this problem would be regular sampling, at the cost of aliasing artifacts. To prevent these, we opted for a custom semi-stochastic sampling procedure, including a connection sample, whenever a warp was too large. Deep Point Correlation Design allows us to combine regularity and stochasticity in a mix-and-match fashion. The best power spectrum to aim for would likely be a convex combination of a blue-noise pattern like BNOT and regular sampling.

For Laplacian Kernel Splatting, we require sampling in the space of PSFs and a (initial) sampling of spreadlets within each PSF. Both tasks require a non-stationary point pattern. A corresponding avenue for future work is discussed in Sec. 7.3. Finally, Stereo Cue Fusion could benefit from Deep Point Correlation Design in the prior extraction step, where a stratification of samples is required.

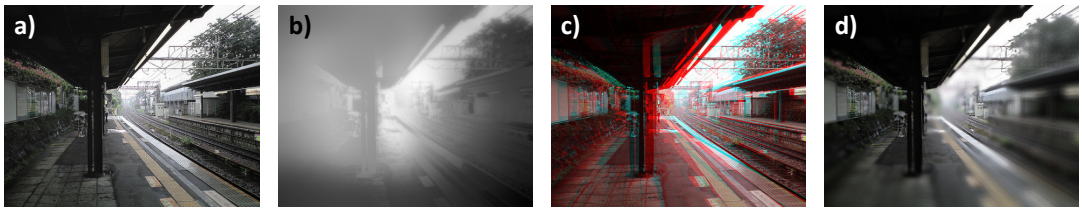


FIGURE 7.1: *Perceptually plausible disparity does not provide high-quality depth-of-field.* (a) *Original monoscopic image.* (b) *Perceptually plausible disparity created with the method developed in Chapter 6.* (c) *Using* b) *to create a stereoscopic image pair from* a). (d) *Using* b) *to apply depth of field to* a). *Even though stereoscopic appearance is acceptable, the depth-of-field effect suffers from a heavy distortion of the focal plane.*

A seemingly promising combination of ideas would be the use of perceptually plausible disparity as produced by Stereo Cue Fusion as a depth input for distribution effect rendering. Anecdotal experiments suggest, however, that the physical inaccuracies of plausible disparity are too strong to be acceptable for a depth-of-field effect (Fig. 7.1). This is not surprising, as the disparities were carefully designed for the very specific use case of 2D-to-3D conversion.

7.3 Future Work

In this section we elaborate on possible directions for future work. First, separate deliberations for the individual chapters of this thesis are given, before engaging in a more general discussion on future avenues for novel-view synthesis.

Deep Point Correlation Design The exact scope of our method remains mostly unclear. Ultimately we would want to ask if *any* point pattern can be learned as we have only shown a small but important subset. This would require a large-scale analysis, since our agendas facilitate many degrees of freedom, including point count, dimensionality, correlation descriptors, projections, and distance metrics. An important application of our method is Monte Carlo integration. Future work will need to perform a detailed convergence analysis for different agendas.

For Monte Carlo integation, enforcing correlations across non-canonical axes has been shown to reduce integration error [Singh and Jarosz, 2017]. So far, our learned mapping does only support axis-aligned projections, but an extension to the general case appears realizable.

Point patterns of spatially-varying density would extend the scope of our method significantly. This could be realized by acknowledging the observation of Wei and Wang [2011] that working in the differential domain allows for spatial adaptivity. Since our filters work on differentials as well, a simple density-dependent scaling of the distances involved in our filter design could be a solution. A naïve implementation would probably not be able to adapt to high-frequency spatial changes in density. Here we could exploit the recursive structure of our model, by progressively adding higher frequencies to the target densities over the course of the filter sequence. Similarly, distances could be scaled as a function of angle to realize certain kinds of anisotropies.

Another desirable property of point patterns is their applicability on manifolds, like e. g., surface meshes. This could be realized similarly to our gridding approach: After each filtering step, the points could be projected back to the closest location on

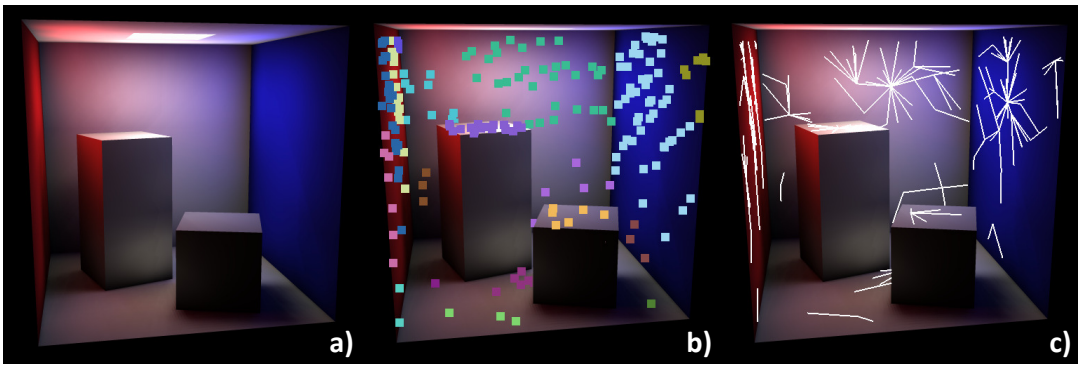


FIGURE 7.2: *Minimal Warping for global illumination.* (a) *A scene illuminated using instant radiosity.* (b) *VPL clusters (colors indicate cluster affiliation).* (c) *Minimal-warping trees could be used to warp VPL shadows.*

the manifold. Again, a certain kind of scale-space progressivity over the course of the filter sequence might help with handling high frequencies of the target geometry.

Minimal Warping It would be interesting to investigate other possible input formats to our method. Per-pixel lists of shading samples or point clouds [Nalbach et al., 2014] are potential candidates amenable to our approach.

The fact that our method is able to handle soft shadows bears the assumption that it could also be used to simulate other global-illumination effects beyond caustics. In particular, a many-lights approach [Keller, 1997] appears promising. The shadows cast by adjacent virtual point lights (VPLs) are highly similar to each other. In a first step, VPLs could be clustered based on position and corresponding surface normal (Fig. 7.2, b). Then, a minimal-warping tree could be created for each cluster (Fig. 7.2, c). This way, shadow rays would only have to be traced for one cluster representative, while all other VPL shadows could be produced by our efficient warping method. Regrettably, this procedure cannot be offhandedly combined with light cuts [Walter et al., 2005] or other hierarchical acceleration techniques, since the view tree always considers entire images. However, a custom locality-aware procedure based on the tiling approach developed in Sec. 4.4.2 is conceivable.

Laplacian Kernel Splatting The biggest limitation of this approach is the restriction to a low number of parameter dimensions. For example, we approximate object motion using linear segments, mostly because of the dimensionality restriction imposed by the pre-calculation. Also, the ability to change the appearance of depth-of-field in a dynamical optical system is desirable.

This problem could be attacked in two ways: On the one hand, a more efficient PSF creation and sparsification pipeline could be developed, as the limiting factor is pre-computation *time*. On the other hand, a more aggressive sub-sampling in the space of PSFs could be achieved by a custom PSF interpolation scheme. This interpolation, however, would have to be performed in the sparse spreadlet domain.

Stereo Cue Fusion In future work we would like to integrate more sophisticated cues into our method. Structure-from-motion could be introduced into our system as a cue itself. More elaborate priors conditioned on texture and flow could add to the inference without imposing additional complexity and compute cost. We also would like to model cue fusion with the goal of improving the quality of stereoscopic

experience when binocular disparity is given, instead of producing it from monocular images. Finally, our fusion is not limited to inference of disparity, but could include other modalities such as observer motion, multiple images or real-time sensor data.

This thesis has injected "islands" of intelligence into the image-based rendering pipeline. However, considering recent developments in machine learning, end-to-end trainable pipelines tend to outperform most alternatives. Promising future work could directly apply this line of thought to the algorithms developed in this thesis. Point patterns can be optimized for a specific use case, like Monte Carlo integration in a domain of interest. In the context of our Deep Point Correlation Design, only the agenda would need to change to incorporate task-specific losses. Likewise, Laplacian Kernel Splatting could benefit from an end-to-end approach. While the current system relies on hard-coded mathematics to integrate a sparse PSF representation, a trainable densification scheme would probably achieve higher compression rates and efficiency. That way, our system would become an auto-encoder [Hinton and Salakhutdinov, 2006] for PSFs. Stereo Cue Fusion provides back-propagatable modules that can complement a fully trainable deep system.

Ultimately, we would want all components of the view synthesis pipeline to be aware of each other. That way, the sampler in a stochastic path tracer would be able to adapt to the demands of a stereo conversion system after large-scale depth of field has been reconstructed from sparse light field probes. Current state of the art comes closer in achieving this goal by using back-propagatable modules throughout the rendering pipeline. Besides technical challenges like controlling the gradient magnitude, there are fundamental problems that need to be solved. For example, a differentiable renderer needs to be able to back-propagate through binary visibility functions [Li et al., 2018].

Taking these ideas a step further, it appears that dedicated modules in the rendering pipeline may not be necessary at all. Recently, Eslami et al. [2018] have developed a system that consumes a small number of images from a scene. Their generative model solely consists of two neural networks and allows to walk through a three-dimensional reconstruction of the scene without ever being explicitly programmed to do so.

7.4 Closing Remarks

The focus of this thesis was to make image-based novel-view synthesis more efficient and versatile. Along with the considerations and examples developed in this chapter, it serves as a strong indicator that artificial intelligence is a valuable tool for view synthesis tasks. We believe and have shown evidence that an informed combination of artificial intelligence, massively parallel algorithms, and perception-driven approximations has the potential to pave the way towards photo-realistic imagery in dynamic and interactive virtual environments.

Bibliography - Own Work

Kaplanyan, Anton, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo [2019]. "DeepFovea: Neural Reconstruction for Foveated Rendering and Video Compression using Learned Statistics of Natural Videos". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*.

Kellnhofer, Petr, Thomas Leimkühler, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel [2015]. "What Makes 2D-to-3D Stereo Conversion Perceptually Plausible?" In: *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception*. SAP '15. Tübingen, Germany: ACM, pp. 59–66.

Leimkühler, Thomas, Petr Kellnhofer, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel [2016]. "Perceptual real-time 2D-to-3D conversion using cue fusion". In: *Proc. Graphics Interface*.

Leimkühler, Thomas, Hans-Peter Seidel, and Tobias Ritschel [2017]. "Minimal Warping: Planning Incremental Novel-view Synthesis". In: *Computer Graphics Forum (Proc. EGSR)* 36.4.

Leimkühler, Thomas, Hans-Peter Seidel, and Tobias Ritschel [2018a]. "Laplacian Kernel Splatting for Efficient Depth-of-field and Motion Blur Synthesis or Reconstruction". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 37.4.

Leimkühler, Thomas, Petr Kellnhofer, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel [2018b]. "Perceptual real-time 2D-to-3D conversion using cue fusion". In: *IEEE Transactions on Visualization and Computer Graphics* 24.6, pp. 2037–2050.

Leimkühler, Thomas, Gurprit Singh, Karol Myszkowski, Hans-Peter Seidel, and Tobias Ritschel [2019]. "Deep Point Correlation Design". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*.

Bibliography

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng [2016]. "TensorFlow: A System for Large-scale Machine Learning". In: *OSDI*.

Adelson, Edward H and James R Bergen [1991]. "The plenoptic function and the elements of early vision". In: *Computational Models of Visual Processing*.

Ahmed, Abdalla GM, Hélène Perrier, David Coeurjolly, Victor Ostromoukhov, Jianwei Guo, Dong-Ming Yan, Hui Huang, and Oliver Deussen [2016]. "Low-discrepancy blue noise sampling". In: *ACM Trans. Graph.* 35.6.

Akenine-Möller, Tomas, Jacob Munkberg, and Jon Hasselgren [2007]. "Stochastic rasterization using time-continuous triangles". In: *Graphics Hardware*, pp. 7–16.

Anderson, Luke, Tzu-Mao Li, Jaakko Lehtinen, and Frédo Durand [2017]. "Aether: An Embedded Domain Specific Sampling Language for Monte Carlo Rendering". In: *ACM Trans. Graph.* 36.4.

Assa, Jackie and Lior Wolf [2007]. "Diorama construction from single images". In: *Comp. Graph. Forum (Proc. EG)* 26.3, pp. 599–608.

Bach, Francis, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski [2011]. "Optimization with Sparsity-Inducing Penalties". In: *arxiv:1108.0775*.

Bailey, Donald G [2003]. "Sub-pixel estimation of local extrema". In: *Proc. Image and Vision Computing*, pp. 414–19.

Balzer, Michael, Thomas Schlömer, and Oliver Deussen [2009]. "Capacity-constrained point distributions: a variant of Lloyd's method". In: *ACM Trans. Graph.* 28.3.

Barlas, Gerassimos [2014]. *Multicore and GPU Programming: An integrated approach*. Elsevier.

Barnard, Stephen T. [1983]. "Interpreting perspective images". In: *Artificial Intelligence* 21.4, pp. 435–462.

Barron, Jonathan T and Ben Poole [2016]. "The fast bilateral solver". In: *European Conference on Computer Vision*. Springer, pp. 617–632.

Baum, Seth [2017]. "A survey of artificial general intelligence projects for ethics, risk, and policy". In: *Global Catastrophic Risk Institute Working Paper*, pp. 17–1.

Belcour, Laurent, Cyril Soler, Kartic Subr, Nicolas Holzschuch, and Fredo Durand [2013]. "5D Covariance tracing for efficient defocus and motion blur". In: *ACM Trans. Graph (Proc. SIGGRAPH)* 32.3, p. 31.

Bernstein, Gilbert Louis, Chinmayee Shah, Crystal Lemire, Zachary Devito, Matthew Fisher, Philip Levis, and Pat Hanrahan [2016]. "Ebb: A DSL for Physical Simulation on CPUs and GPUs". In: *ACM Trans. Graph.* 35.2.

Bhat, Pravin, Brian Curless, Michael Cohen, and C Zitnick [2008]. "Fourier analysis of the 2D screened Poisson equation for gradient domain problems". In: *ECCV*, pp. 114–28.

Bhat, Pravin, C Lawrence Zitnick, Michael Cohen, and Brian Curless [2010]. "Gradientshop: A gradient-domain optimization framework for image and video filtering". In: *ACM Trans. Graph.* 29.2.

Bowers, John, Rui Wang, Li-Yi Wei, and David Maletz [2010]. "Parallel Poisson disk sampling with spectrum analysis on surfaces". In: *ACM Trans. Graph.* 29.6.

Bowles, Huw, Kenny Mitchell, Robert W. Sumner, Jeremy Moore, and Markus Gross [2012]. "Iterative Image Warping". In: *Computer Graphics Forum* 31.2, pp. 237–246. ISSN: 1467-8659.

Bresnahan, Timothy F. and M. Trajtenberg [1995]. "General purpose technologies 'Engines of growth'?" In: *Journal of Econometrics* 65.1, pp. 83 –108.

Chaitanya, Chakravarty R. Alla, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila [2017]. "Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder". In: *ACM Trans. Graph.* 36.4.

Chen, Jiawen, Sylvain Paris, and Frédo Durand [2007]. "Real-time edge-aware image processing with the bilateral grid". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 26.3, p. 103.

Chen, Shenchang Eric and Lance Williams [1993]. "View Interpolation for Image Synthesis". In: *SIGGRAPH*, pp. 279–88.

Chen, Zhonggui, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang [2012]. "Variational blue noise sampling". In: *IEEE Trans. Vis Comp. Graph.* 18.10, pp. 1784–96.

Cheng, Chao-Chung, Chung-Te Li, and Liang-Gee Chen [2010]. "An Ultra-Low-Cost 2D-to-3D Video Conversion System". In: *SID* 41.1, pp. 766–9.

Chiu, Kenneth, Peter Shirley, and Changyaw Wang [1994]. "Graphics Gems IV". In: ed. by Paul S. Heckbert. Chap. Multi-jittered Sampling, pp. 370–374.

Christensen, Per, Andrew Kensler, and Charlie Kilpatrick [2018]. "Progressive Multi-Jittered Sample Sequences". In: *Comp. Graph. Forum (Proc.s of EGSR)* 37.4.

Cleary, Anne, Denis Connolly, and Neil McKenzie [2014]. *Metaperceptual Helmets*. URL: `http://www.connolly-cleary.com/Home/helmets.html` [visited on 03/14/2019].

Cook, Robert L [1986]. "Stochastic sampling in computer graphics". In: *ACM Trans. Graph.* 5.1, pp. 51–72.

Cook, Robert L., Thomas Porter, and Loren Carpenter [Jan. 1984]. "Distributed Ray Tracing". In: *SIGGRAPH Comput. Graph.* 18.3, pp. 137–45.

Cook, Robert L., Loren Carpenter, and Edwin Catmull [1987]. "The Reyes Image Rendering Architecture". In: *SIGGRAPH Comp. Graph.* 21.4, pp. 95–102.

Cozman, F. and E. Krotkov [1997]. "Depth from scattering". In: *CVPR*, pp. 801–806.

Crow, Franklin C [1984]. "Summed-area tables for texture mapping". In: *ACM SIGGRAPH Computer Graphics* 18.3, pp. 207–12.

Csáji, Balázs Csanád [2001]. "Approximation with artificial neural networks". In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24, p. 48.

Dabov, Kostadin, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian [2006]. "Image denoising with block-matching and 3D filtering". In: *Proc. SPIE*. Vol. 6064. 30.

Dahm, Ken and Alexander Keller [2017]. "Learning Light Transport the Reinforced Way". In: *CoRR* abs/1701.07403.

Dammertz, Sabrina and Alexander Keller [2008]. "Image synthesis by rank-1 lattices". In: *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer, pp. 217–236.

Darrell, T. and K. Wohn [1988]. "Pyramid based depth from focus". In: *CVPR*, pp. 504–509.

De Goes, Fernando, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun [2012]. "Blue noise through optimal transport". In: *ACM Trans. Graph.* 31.6.

Devito, Zachary, Michael Mara, Michael Zollhöfer, Gilbert Bernstein, Jonathan Ragan-Kelley, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Niessner [2017]. "Opt: A Domain Specific Language for Non-Linear Least Squares Optimization in Graphics and Imaging". In: *ACM Trans. Graph.* 36.5.

Didyk, Piotr, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel [2010a]. "Adaptive Image-space Stereo View Synthesis". In: *Proc. VMV*, pp. 299–306.

Didyk, Piotr, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel [2010b]. "Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays". In: *Comp. Graph. Forum (Proc. Eurographics)* 29.2, pp. 713–22.

Didyk, Piotr, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, Hans-Peter Seidel, and Wojciech Matusik [2012]. "A Luminance-Contrast-Aware Disparity Model and Applications". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 31.6.

Didyk, Piotr, Pitchaya Sitthi-Amorn, William T. Freeman, Frédo Durand, and Wojciech Matusik [2013]. "Joint View Expansion and Filtering for Automultiscopic 3D Displays". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 32.6.

Dobkin, David P., David Eppstein, and Don P. Mitchell [Oct. 1996]. "Computing the Discrepancy with Applications to Supersampling Patterns". In: *ACM Trans. Graph.* 15.4, pp. 354–76.

Dong, Zhao, Thorsten Grosch, Tobias Ritschel, Jan Kautz, and Hans-Peter Seidel [2009]. "Real-time Indirect Illumination with Clustered Visibility". In: *VMV*, pp. 187–196.

Donoho, David L, Iain M Johnstone, Jeffrey C Hoch, and Alan S Stern [1992]. "Maximum entropy and the nearly black object". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 54.1, pp. 41–67.

Durand, Frédo, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X Sillion [2005]. "A frequency analysis of light transport". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 24.3, pp. 1115–26.

Dutré, Philip, Eric P Lafortune, and Yves Willems [1993]. "Monte Carlo light tracing with direct computation of pixel intensities". In: *Proc. Computational Graphics and Visualisation Techniques*, pp. 128–37.

Egan, Kevin, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi [2009]. "Frequency analysis and sheared reconstruction for rendering motion blur". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 28.3.

Eigen, David, Christian Puhrsch, and Rob Fergus [2014]. "Depth map prediction from a single image using a multi-scale deep network". In: *Advances in neural information processing systems*, pp. 2366–2374.

Elek, Oskar, Pablo Bauszat, Tobias Ritschel, Marcus Magnor, and Hans-Peter Seidel [2014]. "Spectral Ray Differentials". In: *Proc. EGSR* 33.4.

Eslami, SM Ali, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. [2018]. "Neural scene representation and rendering". In: *Science* 360.6394, pp. 1204–1210.

Farbman, Zeev, Raanan Fattal, and Dani Lischinski [2011]. "Convolution pyramids". In: *ACM Trans. Graph.* 30.6, pp. 175–1.

Fattal, Raanan [Aug. 2008]. "Single Image Dehazing". In: *ACM Trans. Graph.* 27.3, 72:1–72:9.

Fattal, Raanan [2011]. "Blue-noise point sampling using kernel density model". In: *ACM Trans. Graph.* 30.4.

Filippini, Heather R and Martin S Banks [2009]. "Limits of stereopsis explained by local cross-correlation". In: *J Vis.* 9.1.

Flavell, John H, Barbara A Everett, Karen Croft, and Eleanor R Flavell [1981]. "Young children's knowledge about visual perception: Further evidence for the Level 1–Level 2 distinction." In: *Developmental psychology* 17.1, p. 99.

Fleming, Roland W, Ron O Dror, and Edward H Adelson [2003]. "Real-world illumination and the perception of surface reflectance properties". In: *Journal of vision* 3.5, pp. 3–3.

Flynn, Michael J [1972]. "Some computer organizations and their effectiveness". In: *IEEE transactions on computers* 100.9, pp. 948–960.

Galić, Irena, Joachim Weickert, Martin Welk, Andrés Bruhn, Alexander Belyaev, and Hans-Peter Seidel [2008]. "Image compression with anisotropic diffusion". In: *J Math. Imaging and Vision* 31.2, pp. 255–69.

Garg, Ravi, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid [2016]. "Unsupervised CNN for single view depth estimation: Geometry to the rescue". In: *European Conference on Computer Vision*. Springer, pp. 740–756.

Georgiev, Iliyan and Marcos Fajardo [2016]. "Blue-noise Dithered Sampling". In: *ACM SIGGRAPH 2016 Talks*.

Gerace, Adam, Andrew Day, Sharon Casey, and Philip Mohr [2013]. "An exploratory investigation of the process of perspective taking in interpersonal situations". In: *Journal of Relationships Research* 4.

Gershun, Andreī [1939]. "The light field". In: *Journal of Mathematics and Physics* 18.1-4, pp. 51–151.

Gibson, K. B., S. J. Belongie, and T. Q. Nguyen [2013]. "Example based depth from fog". In: *Proc. ICIP*, pp. 728–32.

Göransson, Jhonny and Andreas Karlsson [2007]. "Practical post-process depth of field". In: *GPU Gems 3*, pp. 583–606.

Gortler, Steven J, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen [1996]. "The lumigraph". In: *Siggraph*. Vol. 96. 30, pp. 43–54.

Goy, Michael [2013]. *American cinematographer manual*. Vol. 10. Am. Cinemat.

Grant, Adam M and James W Berry [2011]. "The necessity of others is the mother of invention: Intrinsic and prosocial motivations, perspective taking, and creativity". In: *Academy of management journal* 54.1, pp. 73–96.

Green, Peter J [1984]. "Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives". In: *J Royal Stat. Soc. B*, pp. 149–92.

Guttmann, M., L. Wolf, and D. Cohen-Or [2009]. "Semi-automatic stereo extraction from video footage". In: *Proc. ICCV*.

Haeberli, Paul and Kurt Akeley [1990]. "The accumulation buffer: hardware support for high-quality rendering". In: *SIGGRAPH Comp. Graph.* 24.4, pp. 309–18.

Halton, John H [1964]. "Algorithm 247: Radical-inverse quasi-random point sequence". In: *Communications of the ACM* 7.12, pp. 701–702.

Harris, Julie M and Laurie M Wilcox [2009]. "The role of monocularly visible regions in depth and surface perception". In: *Vision research* 49.22, pp. 2666–2685.

Hasselgren, Jon and Tomas Akenine-Möller [2006]. "An efficient multi-view rasterization architecture". In: *Proc. EGSR*, pp. 61–72.

Heck, Daniel, Thomas Schlömer, and Oliver Deussen [2013]. "Blue noise sampling with controlled aliasing". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.3, p. 25.

Heckbert, Paul S. [1986]. "Filtering by Repeated Integration". In: *Proc. SIGGRAPH*. ACM, pp. 315–321. ISBN: 0-89791-196-2.

Heide, F., G. Wetzstein, R. Raskar, and W. Heidrich [2013]. "Adaptive Image Synthesis for Compressive Displays". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.4, pp. 1–11.

Heide, Felix, Steven Diamond, Matthias Niessner, Jonathan Ragan-Kelley, Wolfgang Heidrich, and Gordon Wetzstein [2016]. "ProxImaL: Efficient Image Optimization Using Proximal Algorithms". In: *ACM Trans. Graph.* 35.4.

Hermosilla, Pedro, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski [2018]. "Monte Carlo Convolution for Learning on Non-uniformly Sampled Point Clouds". In: *ACM Trans. Graph (Proc. SIGGRAPH Asia)* 37.5.

Hicks, JS and RF Wheeling [1959]. "An efficient method for generating uniformly distributed points on the surface of an n-dimensional sphere". In: *Comm. ACM* 2.4, pp. 17–19.

Hinton, Geoffrey E and Ruslan R Salakhutdinov [2006]. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507.

Hoiem, Derek, Alexei A. Efros, and Martial Hebert [2005]. "Automatic photo pop-up". In: *ACM Trans. Graph.* 24.3, pp. 577–584.

Howard, I.P. and B.J. Rogers [2012]. *Perceiving in Depth*. Oxford Psychology Series.

Huang, Xiaojun, Lianghao Wang, Junjun Huang, Dongxiao Li, and Ming Zhang [2009]. "A Depth Extraction Method Based on Motion and Geometry for 2D to 3D Conversion". In: *Proc. IITA*. Nanchang, China, pp. 294–298.

Hullin, Matthias, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee [2011]. "Physically-based real-time lens flare rendering". In: *ACM Trans Graph. (Proc. SIGGRAPH Asia)* 30.4, p. 108.

Igehy, Homan [1999]. "Tracing ray differentials". In: *Proc. SIGGRAPH*. ACM, pp. 179–186.

Itakura, Shoji [2004]. "Gaze-following and joint visual attention in nonhuman animals". In: *Japanese Psychological Research* 46.3, pp. 216–226.

Ito, Atsushi, Salil Tambe, Kaushik Mitra, Aswin C Sankaranarayanan, and Ashok Veeraraghavan [2014]. "Compressive epsilon photography for post-capture control in digital imaging". In: *ACM Trans. Graph.* 33.4, p. 88.

Jiang, Min, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang [2015]. "Blue noise sampling using an SPH-based method". In: *ACM Trans. Graph.* 34.6.

Kailkhura, Bhavya, Jayaraman J Thiagarajan, Peer-Timo Bremer, and Pramod K Varshney [2016]. "Stair blue noise sampling". In: *ACM Trans. Graph.* 35.6.

Kajiya, James T [1986]. "The rendering equation". In: *ACM SIGGRAPH computer graphics*. Vol. 20. 4. ACM, pp. 143–150.

Kalantari, Nima Khademi and Pradeep Sen [2013]. "Removing the noise in Monte Carlo rendering with general image denoising algorithms". In: *Comp. Graph. Forum* 32.2, pp. 93–102.

Kallweit, Simon, Thomas Müller, Brian Mcwilliams, Markus Gross, and Jan Novák [2017]. "Deep Scattering: Rendering Atmospheric Clouds with Radiance-predicting Neural Networks". In: *ACM Trans. Graph.* 36.6.

Kane, D., P. Guan, and M.S. Banks [2014]. "The Limits of Human Stereopsis in Space and Time". In: *J Neurosc.* 34.4, pp. 1397–408.

Karsch, K., Ce Liu, and Sing Bing Kang [2014]. "Depth Transfer: Depth Extraction from Video Using Non-Parametric Sampling". In: *IEEE PAMI* 36.11.

Keller, Alexander [1997]. "Instant Radiosity". In: *Proc. SIGGRAPH 1997*. ACM Press/Addison-Wesley Publishing Co., pp. 49–56.

Keller, Alexander [2006]. "Myths of computer graphics". In: *Monte Carlo and Quasi-Monte Carlo Methods 2004*. Springer, pp. 217–243.

Kellnhofer, Petr, Piotr Didyk, Tobias Ritschel, Belen Masia, Karol Myszkowski, and Hans-Peter Seidel [2016]. "Motion Parallax in Stereo 3D: Model and Applications". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2016)* 35.6.

Kensler, Andrew [2013]. *Correlated multi-jittered sampling*. Tech. rep. Pixar Technical Memo.

Kettunen, Markus, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker [2015]. "Gradient-domain path tracing". In: *ACM Trans. Graph.* 34.4, p. 123.

Kirk, David [2007]. "NVIDIA CUDA software and GPU parallel computing architecture". In: *ISMM*. Vol. 7, pp. 103–104.

Knill, David C and Whitman Richards [1996]. *Perception as Bayesian inference*. Cambridge University Press.

Knutsson, Hans and C-F Westin [1993]. "Normalized and differential convolution". In: *CVPR*, pp. 515–23.

Kolb, Craig, Don Mitchell, and Pat Hanrahan [1995]. "A realistic camera model for computer graphics". In: *SIGGRAPH*, pp. 317–24.

Koller, Daphne and Nir Friedman [2009]. *Probabilistic graphical models: Principles and techniques*. MIT press.

Konrad, Janusz, Meng Wang, and Prakash Ishwar [2012]. "2D-to-3D image conversion by learning depth from examples". In: *CVPR*, pp. 16–22.

Kontkanen, Janne, Jussi Räsänen, and Alexander Keller [2006]. "Irradiance filtering for Monte Carlo ray tracing". In: *Proc. MC QMC*, pp. 259–272.

Kopf, Johannes, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski [2006]. "Recursive Wang tiles for real-time blue noise". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 25.3.

Kopf, Johannes, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele [2007]. "Joint Bilateral Upsampling". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 26.3.

Kopf, Johannes, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele [2013]. "Image-based rendering in the gradient domain". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.6.

Krähenbühl, Philipp and Vladlen Koltun [2011]. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *NIPS*, pp. 109–117.

Kraus, Martin and Magnus Strengert [2007]. "Depth-of-Field Rendering by Pyramidal Image Processing". In: *Comp. Graph Forum*. Vol. 26. 3, pp. 645–54.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton [2012]. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*, pp. 1097–105.

Kulla, Christopher, Alejandro Conty, Clifford Stein, and Larry Gritz [Aug. 2018]. "Sony Pictures Imageworks Arnold". In: *ACM Trans. Graph.* 37.3, 29:1–29:18. ISSN: 0730-0301.

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira [2001]. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proc. ICML*, pp. 282–289.

Lagae, Ares and Philip Dutre [2008]. "A Comparison of Methods for Generating Poisson Disk Distributions". In: *Computer Graphics Forum* 27.1, pp. 114–129.

Landy, Michael S., Laurence T. Maloney, Elizabeth B. Johnston, and Mark Young [1995]. "Measurement and modeling of depth cue combination: In defense of weak fusion". In: *Vis. Res.* 35.3, pp. 389–412.

Lang, Manuel, Alexander Hornung, Oliver Wang, Steven Poulakos, Aljoscha Smolic, and Markus Gross [2010]. "Nonlinear disparity mapping for stereoscopic 3D". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 29.4.

Lang, Manuel, Oliver Wang, Tunc Aydin, Aljoscha Smolic, and Markus Gross [2012]. "Practical temporal consistency for image-based graphics applications". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.4.

Lee, Sungkil, Gerard Jounghyun Kim, and Seungmoon Choi [2008]. "Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers". In: *Comp. Graph. Forum* 27.7, pp. 1955–62.

Lee, Sungkil, Elmar Eisemann, and Hans-Peter Seidel [2009]. "Depth-of-field rendering with multiview synthesis". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*. Vol. 28. 5. ACM, p. 134.

Lee, Sungkil, Elmar Eisemann, and Hans-Peter Seidel [2010]. "Real-time lens blur effects and focus control". In: *ACM Trans. Graph (Proc. SIGGRAPH)* 29.4.

Lehtinen, Jaakko, Timo Aila, Jiawen Chen, Samuli Laine, and Frédo Durand [2011]. "Temporal Light Field Reconstruction for Rendering Distribution Effects". In: *ACM Trans. Graph.* 30.4.

Lehtinen, Jaakko, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila [2013]. "Gradient-domain metropolis light transport". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 32.4, p. 95.

Li, Bo, Chunhua Shen, Yuchao Dai, Anton Van Den Hengel, and Mingyi He [2015]. "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs". In: *Proc. CVPR*, pp. 1119–1127.

Li, Tzu-Mao, Miika Aittala, Frédo Durand, and Jaakko Lehtinen [2018]. "Differentiable Monte Carlo Ray Tracing through Edge Sampling". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6, 222:1–222:11.

Lin, Jingyu, Xiangyang Ji, Wenli Xu, and Qionghai Dai [2013]. "Absolute Depth Estimation From a Single Defocused Image". In: *IEEE Trans. Image Processing* 22.11, pp. 4545–4550.

Linnainmaa, Seppo [1970]. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". In: *Master's Thesis, Univ. Helsinki*.

Lippman, Andrew [1980]. "Movie-maps: An application of the optical videodisc to computer graphics". In: *ACM SIGGRAPH*. Vol. 14. 3, pp. 32–42.

Lischinski, Dani and Ari Rappoport [1998]. "Image-based rendering for non-diffuse synthetic scenes". In: *Rendering Techniques' 98*. Springer, pp. 301–314.

Liu, Fayao, Chunhua Shen, Guosheng Lin, and Ian Reid [2016]. "Learning depth from single monocular images using deep convolutional neural fields". In: *IEEE transactions on pattern analysis and machine intelligence* 38.10, pp. 2024–2039.

Liu, Xueting, Xiangyu Mao, Xuan Yang, Linling Zhang, and Tien-Tsin Wong [2013]. "Stereoscopizing cel animations". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 32.6, p. 223.

Lloyd, Stuart [1982]. "Least squares quantization in PCM". In: *IEEE Trans. Inf. Theory* 28.2, pp. 129–37.

Lochmann, Gerrit, Bernhard, Tobias Ritschel, Stefan Müller, and Hans-Peter Seidel [2014]. "Real-time Reflective and Refractive Novel-view Synthesis". In: *Proc. VMV*, pp. 9–16.

Lucas, Bruce D and Takeo Kanade [1981]. "An iterative image registration technique with an application to stereo vision." In: *IJCAI* 81, pp. 74–679.

Ma, Wei-Ying and B.S. Manjunath [2000]. "EdgeFlow: A technique for boundary detection and image segmentation". In: *IEEE Trans. Image Processing* 9.8, pp. 1375–1388.

Mark, William R., Leonard McMillan, and Gary Bishop [1997]. "Post-rendering 3D Warping". In: *Proc. i3D*.

Marlow, Phillip J, Juno Kim, and Barton L Anderson [2012]. "The perception and misperception of specular surface reflectance". In: *Current Biology* 22.20, pp. 1909–1913.

Maximov, Maxim, Tobias Ritschel, and Mario Fritz [2018]. "Deep Appearance Maps". In: *arXiv preprint arXiv:1804.00863*.

McCool, Michael and Eugene Fiume [1992]. "Hierarchical Poisson disk sampling distributions". In: *Proc. Graphics interface*. Vol. 92, pp. 94–105.

McCool, Michael D [1999]. "Anisotropic diffusion for Monte Carlo noise reduction". In: *ACM Trans. Graph.* 18.2, pp. 171–94.

McCorduck, Pamela [2009]. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. AK Peters/CRC Press.

McGuire, Morgan, Padraic Hennessy, Michael Bukowski, and Brian Osman [2012]. "A reconstruction filter for plausible motion blur". In: *i3D*, pp. 135–42.

McMillan, Leonard [1997]. "An Image-Based Approach to Three-Dimensional Computer Graphics". PhD thesis. University of North Carolina at Chapel Hill.

Merkle, P., Y. Morvan, A. Smolic, D. Farin, K. Müller, P. H. N. de With, and T. Wiegand [2009]. "The effects of multiview depth video compression on multiview rendering". In: *Signal Processing: Im. Commun.* 24.1-2.

Michaelis, M. and G. Sommer [1994]. "Junction classification by multiple orientation detection". English. In: *Proc. ECCV*, pp. 101–8.

Miksik, Ondrej, Daniel Munoz, J Andrew Bagnell, and Martial Hebert [2013]. "Efficient temporal consistency for streaming video scene analysis". In: *Proc. ICRA*, pp. 133–9.

Mitchell, Don P. [1992]. "Ray Tracing and Irregularities of Distribution". In: *In Third Eurographics Workshop on Rendering*, pp. 61–69.

Mitchell, Scott A., Mohamed S. Ebeida, Muhammad A. Awad, Chonhyon Park, Anjul Patney, Ahmad A. Rushdi, Laura P. Swiler, Dinesh Manocha, and Li-Yi Wei [2018]. "Spoke-Darts for High-Dimensional Blue-Noise Sampling". In: *ACM Trans. Graph.* 37.2.

Müller, Thomas, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák [2018]. "Neural Importance Sampling". In: *arXiv:1808.03856*.

Mulligan, Jeffrey B and Albert J Ahumada [1992]. "Principled halftoning based on human vision models". In: *Human Vision, Visual Processing, and Digital Display III*. Vol. 1666.

Munkberg, Jacob, Karthik Vaidyanathan, Jon Hasselgren, Petrik Clarberg, and Tomas Akenine-Möller [2014]. "Layered Reconstruction for Defocus and Motion Blur". In: *Comp. Graph. Forum* 33.4, pp. 81–92.

Murata, H., Y. Mori, S. Yamashita, A. Maenaka, S. Okada, K. Oyamada, and S. Kishimoto [1998]. "A Real-Time 2-D to 3-D Image Conversion Technique Using Computed Image Depth". In: *SID* 29.1, pp. 919–23.

Nalbach, Oliver, Tobias Ritschel, and Hans-Peter Seidel [2014]. "Deep Screen Space". In: *I3D*. ACM.

Nalbach, Oliver, Elena Arabadzhiyska, Dushyant Mehta, H-P Seidel, and Tobias Ritschel [2017]. "Deep Shading: Convolutional Neural Networks for Screen Space Shading". In: *Comp. Graph. Forum* 36.4, pp. 65–78.

Oliver, Deussen, Hiller Stefan, Van Overveld Cornelius, and Strothotte Thomas [2001]. "Floating Points: A Method for Computing Stipple Drawings". In: *Computer Graphics Forum* 19.3, pp. 41–50.

Orzan, Alexandrina, Adrien Bousseau, Pascal Barla, Holger Winnemöller, Joëlle Thollot, and David Salesin [2013]. "Diffusion curves: a vector representation for smooth-shaded images". In: *Comm. ACM* 56.7, pp. 101–8.

Ostromoukhov, Victor, Charles Donohue, and Pierre-Marc Jodoin [2004]. "Fast hierarchical importance sampling with blue noise properties". In: *ACM Trans. Graph.* 23.3, pp. 488–95.

Öztireli, A Cengiz and Markus Gross [2012]. "Analysis and synthesis of point distributions based on pair correlation". In: *ACM Trans. Graph.* 31.6.

Penrose, Roger [1999]. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Popular Science Series. OUP Oxford.

Pentland, Alex Paul [1987]. "A New Sense for Depth of Field". In: *IEEE PAMI* 4.

Pérez, Patrick, Michel Gangnet, and Andrew Blake [2003]. "Poisson image editing". In: *ACM Trans. Graph (Proc. SIGGRAPH)* 22.3, pp. 313–18.

Perrier, Hélène, David Coeurjolly, Feng Xie, Matt Pharr, Pat Hanrahan, and Victor Ostromoukhov [2018]. "Sequences with Low-Discrepancy Blue-Noise 2-D Projections". In: *Comp. Graph. Forum (Proc. Eurographics)* 37.2.

Pharr, Matt, Wenzel Jakob, and Greg Humphreys [2016]. *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

Piaget, J and B Inhelder [1969]. "The psychology of the child Basic Books". In: *Inc New York*.

Pilleboue, Adrien, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov [2015]. "Variance analysis for Monte Carlo integration". In: *ACM Trans. Graph.* 34.4.

Potmesil, Michael and Indranil Chakravarty [1981]. "A lens and aperture camera model for synthetic image generation". In: *ACM SIGGRAPH Computer Graphics* 15.3, pp. 297–305.

Premože, Simon, Michael Ashikhmin, Jerry Tessendorf, Ravi Ramamoorthi, and Shree Nayar [2004]. "Practical rendering of multiple scattering effects in participating media". In: *Proc. EGWR*, pp. 363–74.

Qi, Charles R, Hao Su, Kaichun Mo, and Leonidas J Guibas [2017]. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *CVPR*.

Qin, Hongxing, Yi Chen, Jinlong He, and Baoquan Chen [2017]. "Wasserstein Blue Noise Sampling". In: *ACM Trans. Graph.* 36.5.

Ramanarayanan, Ganesh, James Ferwerda, Bruce Walter, and Kavita Bala [2007]. "Visual equivalence: Towards a new standard for image fidelity". In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM, p. 76.

Raskar, Ramesh [2009]. "Computational photography: Epsilon to coded photography". In: *Emerging Trends in Visual Computing*, pp. 238–253.

Reinert, Bernhard, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev [2016]. "Projective Blue-Noise Sampling". In: *Comp. Graph. Forum* 35.1, pp. 285–95.

Ren, Peiran, Yue Dong, Stephen Lin, Xin Tong, and Baining Guo [2015]. "Image Based Relighting Using Neural Networks". In: *ACM Trans. Graph.* 34.

Richardt, Christian, Carsten Stoll, Neil Dodgson, Hans-Peter Seidel, and Christian Theobalt [2012]. "Coherent Spatiotemporal Filterung, Upsampling and Rendering of RGBZ Videos". In: *Comp. Graph. Forum* 31.2.

Robinson, Alan E. and Donald I. A. MacLeod [2013]. "Depth and luminance edges attract". In: *Journal of Vision* 13.11.

Robinson, D and C Atcitty [1999]. "Comparison of quasi-and pseudo-Monte Carlo sampling for reliability and uncertainty analysis". In: *Proc. AIAA Probabilistic Methods*.

Rosado, Gilberto [2007]. "Motion blur as a post-processing effect". In: *GPU gems 3*, pp. 575–81.

Roy, Anirban and Sinisa Todorovic [2016]. "Monocular depth estimation using neural regression forest". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5506–5514.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams [1985]. *Learning internal representations by error propagation*. Tech. rep. California Univ. San Diego La Jolla Inst. for Cognitive Science.

Russell, Stuart, Daniel Dewey, and Max Tegmark [2015]. "Research priorities for robust and beneficial artificial intelligence". In: *Ai Magazine* 36.4, pp. 105–114.

Russell, Stuart J and Peter Norvig [2016]. *Artificial intelligence: A modern approach*. Malaysia; Pearson Education Limited.

Saxena, Ashutosh, Min Sun, and Andrew Y. Ng [2009]. "Make3D: Learning 3D Scene Structure from a Single Still Image". In: *PAMI* 31.5, pp. 824–40.

Schmaltz, Christian, Pascal Gwosdek, Andres Bruhn, and Joachim Weickert [2010]. "Electrostatic Halftoning". In: *Comp. Graph. Forum*.

Scott, David W [1979]. "On optimal and data-based histograms". In: *Biometrika* 66.3, pp. 605–610.

Secord, Adrian [2002]. "Weighted voronoi stippling". In: *Proc. NPAR*, pp. 37–43.

Sen, Pradeep and Soheil Darabi [2012]. "On filtering the noise from the random parameters in Monte Carlo rendering." In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.3, pp. 18–1.

Shade, Jonathan, Steven Gortler, Li-wei He, and Richard Szeliski [1998]. "Layered depth images". In: *Proc. SIGGRAPH*, pp. 231–42.

Shirley, Peter et al. [1991]. "Discrepancy as a quality measure for sample distributions". In: *Proc. Eurographics*, pp. 183–194.

Shreiner, Dave, Graham Sellers, John Kessenich, and Bill Licea-Kane [2013]. *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley.

Sibbing, D., T. Sattler, B. Leibe, and L. Kobbelt [2013]. "SIFT-Realistic Rendering". In: *Proc. 3DV*.

Silberman, Nathan, Derek Hoiem, Pushmeet Kohli, and Rob Fergus [2012]. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV*.

Simard, Patrice, Léon Bottou, Patrick Haffner, and Yann LeCun [1999]. "Boxlets: a fast convolution algorithm for signal processing and neural networks". In: *NIPS*, pp. 571–7.

Singh, Gurprit and Wojciech Jarosz [2017]. "Convergence Analysis for Anisotropic Monte Carlo Sampling Spectra". In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 36.4.

Sobol, Ilya M [1994]. *A primer for the Monte Carlo method*. CRC press.

Soler, Cyril, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion [2009]. "Fourier depth of field". In: *ACM Trans. Graph.* 28.2, p. 18.

Steinberger, Markus [2013]. "Dynamic Resource Scheduling on Graphics Processors". PhD thesis.

Stone, John E, David Gohara, and Guochun Shi [2010]. "OpenCL: A parallel programming standard for heterogeneous computing systems". In: *Computing in science & engineering* 12.3, p. 66.

Stratton, George M [1897]. "Vision without inversion of the retinal image." In: *Psychological review* 4.4, p. 341.

Subr, Kartic and Jan Kautz [2013]. "Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration". In: *ACM Trans. Graph.* 32.

Sun, Xin, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo [2012]. "Diffusion curve textures for resolution independent texture mapping." In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.4, pp. 74–1.

Suykens, Frank and Yves D Willems [2001]. "Path differentials and applications". In: *Proc. EGSR*. Springer, pp. 257–268.

Szeliski, Richard [1990]. "Bayesian modeling of uncertainty in low-level vision". In: *IJCV* 5.3, pp. 271–301.

Szeliski, Richard [2010]. *Computer vision: Algorithms and applications*. Springer.

Tam, Wa James, Carlos Vázquez, and Filippo Speranza [2009]. "Three-dimensional TV: A novel method for generating surrogate depth maps using colour information". In: *Proc. SPIE*.

Tao, Michael W., Sunil Hadap, Jitendra Malik, and Ravi Ramamoorthi [2013]. "Depth from Combining Defocus and Correspondence Using Light-Field Cameras". In: *ICCV*, pp. 673–680.

Tomasi, Carlo and Roberto Manduchi [1998]. "Bilateral filtering for gray and color images". In: *Int Conf. Comp. Vis.* Pp. 839–846.

Torborg, Jay and James T Kajiya [1996]. "Talisman: Commodity realtime 3D graphics for the PC". In: *Proc. SIGGRAPH*, pp. 353–63.

Torralba, Antonio [2009]. "How many pixels make an image?" In: *Visual Neuroscience* 26 [01], pp. 123–131.

Ulichney, Robert A [1988]. "Dithering with blue noise". In: *Proc. IEEE* 76.1, pp. 56–79.

Vaidyanathan, Karthik, Jacob Munkberg, Petrik Clarberg, and Marco Salvi [2015]. "Layered light field reconstruction for defocus blur". In: *ACM Trans.Graph.* 34.2, p. 23.

Valencia, Sergio Aguirre and Ramon M Rodriguez-Dagnino [2003]. "Synthesizing stereo 3D views from focus cues in monoscopic 2D images". In: *Electronic Imaging 2003*, pp. 377–388.

Van Waveren, JMP [2016]. "The asynchronous time warp for virtual reality on consumer hardware". In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. ACM, pp. 37–46.

Vedula, Sundar, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade [1999]. "Three-dimensional scene flow". In: *ICCV*. Vol. 2, pp. 722–729.

Vogel, Christoph, Konrad Schindler, and Stefan Roth [2015]. "3D Scene Flow Estimation with a Piecewise Rigid Scene Model". In: *Int. J Comp. Vis.* 115.1, pp. 1–28.

Wachtel, Florent, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando De Goes, Mathieu Desbrun, and Victor Ostromoukhov [2014]. "Fast tile-based adaptive sampling with user-specified Fourier spectra". In: *ACM Trans. Graph.* 33.4.

Walter, Bruce, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P Greenberg [2005]. "Lightcuts: a scalable approach to illumination". In: *ACM Transactions on graphics (TOG)*. Vol. 24. 3. ACM, pp. 1098–1107.

Wandell, Brian A. [1995]. *Foundations of vision*. Sinauer Associates.

Wang, Zhou, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P Simoncelli [2004]. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Trans. Image Proc.* 13.4, pp. 600–612.

Ward, B., Sing Bing Kang, and E.P. Bennett [2011]. "Depth Director: A System for Adding Depth to Movies". In: *IEEE Comp. Graph. and App.* 31.1.

Wei, Li-Yi and Rui Wang [2011]. "Differential domain analysis for non-uniform sampling". In: *ACM Trans. Graph.* 30.4.

Westover, Lee [1989]. "Interactive volume rendering". In: *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*. ACM, pp. 9–16.

Widmer, S, D Pajak, A Schulz, K Pulli, J Kautz, M Goesele, and D Luebke [2015]. "An adaptive acceleration structure for screen-space ray tracing". In: *Proc. HPG*, pp. 67–76.

Yamada, K. and Y. Suzuki [2009]. "Real-time 2D-to-3D conversion at full HD 1080P resolution". In: *IEEE ISCE*, pp. 103–6.

Yan, Dong-Ming, Jian-Wei Guo, Bin Wang, Xiao-Peng Zhang, and Peter Wonka [2015a]. "A survey of blue-noise sampling and its applications". In: *J Comp. Sci. and Tech.* 30.3, pp. 439–52.

Yan, Ling-Qi, Soham Uday Mehta, Ravi Ramamoorthi, and Fredo Durand [2015b]. "Fast 4D Sheared Filtering for Interactive Rendering of Distribution Effects". In: *ACM Trans. Graph.* 35.1, p. 7.

Yang, Lei, Yu-Chiu Tse, Pedro V. Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L. Wilkins [2011]. "Image-based Bidirectional Scene Reprojection". In: *ACM Trans. Graph.* 30.6, 150:1–150:10.

Yang, Zhiyong and Dale Purves [2003]. "A statistical explanation of visual space". In: *Nature Neuroscience* 6.6, pp. 632–640.

Yellott, John I [1983]. "Spectral consequences of photoreceptor sampling in the rhesus retina". In: *Science* 221.4608, pp. 382–5.

Yu, Xuan, Rui Wang, and Jingyi Yu [2010]. "Real-time Depth of Field Rendering via Dynamic Light Field Generation and Filtering". In: *Comp. Graph. Forum*. Vol. 29. 7, pp. 2099–107.

Zhang, Guofeng, Wei Hua, Xueying Qin, Tien-Tsin Wong, and Hujun Bao [2007]. "Stereoscopic video synthesis from a monocular video". In: *IEEE TVCG* 13.4.

Zhang, L., C. Vazquez, and S. Knorr [2011]. "3D-TV Content Creation: Automatic 2D-to-3D Video Conversion". In: *IEEE Trans. Broadcasting* 57.2, pp. 372–83.

Zheng, Quan and Matthias Zwicker [2018]. "Learning to Importance Sample in Primary Sample Space". In: *arxiv:1808.07840*.

Zhou, Kun, Qiming Hou, Zhong Ren, Minmin Gong, Xin Sun, and Baining Guo [2009]. "RenderAnts: interactive REYES rendering on GPUs". In: *ACM Trans. Graph. (Proc. SIGGRAPH)*. Vol. 28. 5, p. 155.

Zhou, Yahan, Haibin Huang, Li-Yi Wei, and Rui Wang [2012]. "Point sampling with general noise spectrum". In: *ACM Trans. Graph.* 31.4, p. 76.

Zhou, Zhi-Hua [2017]. "A brief introduction to weakly supervised learning". In: *National Science Review* 5.1, pp. 44–53.

Zwicker, Matthias, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross [2001]. "Surface splatting". In: *Proc. SIGGRAPH*, pp. 371–8.

Zwicker, Matthias, Wojciech Matusik, Frédo Durand, Hanspeter Pfister, and Clifton Forlines [2006]. "Antialiasing for automultiscopic 3D displays". In: *SIGGRAPH Sketches*.