

Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Masterarbeit Nr. 89

**Example-based Parameter
Retrieval for Procedural Materials
through Abstraction of Visual
Features**

Matti Grüner

Course of Study:	Informatik
Examiner:	Jun.-Prof. Dr.-Ing. Martin Fuchs
Supervisor:	Lena Gieseke, M.F.A.
Commenced:	2015-10-12
Completed:	2016-04-12
CR-Classification:	I.3.7, I.4.7, I.4.8

EXAMPLE-BASED PARAMETER RETRIEVAL FOR PROCEDURAL
MATERIALS THROUGH ABSTRACTION OF VISUAL FEATURES

MATTI GRÜNER

Dedicated to my parents.

ABSTRACT

Designing the surface properties of virtual assets for digital content production is a costly and time-consuming process. As the requirements regarding scene complexity increase to improve visual quality, automatic processes need to be developed to deliver projects on time and on budget. Our contribution solves the problem of finding a procedural displacement map that approximates the surface structure depicted in a reference image captured under unknown lighting conditions. We also estimate the shading parameters needed to provide a fully procedural material based on this unstructured input. The system we propose can automate labor-intensive tasks in look development and has the potential to significantly cut production costs.

ZUSAMMENFASSUNG

Bei der Erstellung von Film- und Medienproduktionen ist die Gestaltung von Materialien für virtuelle Objekte besonders zeit- und kostenintensiv. Mit wachsender Szenenkomplexität werden automatisierte Abläufe immer wichtiger, um Projekte zeitnah und innerhalb eines gesetzten Budgets produzieren zu können. Unsere Methode erstellt prozedurale Oberflächen, die ein gegebenes Material approximieren, das unter unbekanntem Licht aufgenommen wurde. Zusätzlich schätzen wir die Parameter für ein Beleuchtungsmodell, das uns erlaubt das abgebildete Material weiter anzunähern. Unser System kann die arbeitsintensiven Abläufe der Oberflächengestaltung automatisieren und erlaubt damit Einsparungen bei Produktionskosten.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to Jun.-Prof. Dr.-Ing. Martin Fuchs and Lena Gieseke, M.F.A. for their support and valuable suggestions.

I also thank Alina Bopole, Oliver Goroll, Anna Grüner and Tanja Schmidt for proofreading. Oliver has also been a valuable partner in countless assignments in a multitude of lectures, for which I am very thankful.

Last but not least, I would like to thank my friends and family for their love and constant support throughout my studies. I am especially grateful to my girlfriend, Tanja Schmidt, for her constant encouragements.

CONTENTS

I	INTRODUCTION	1
1	MOTIVATION	3
1.1	Problem Statement	5
1.2	Outline	5
II	THEORETICAL FOUNDATIONS	7
2	IMAGES AND COLOR SPACES	9
2.1	Digital Images	9
2.2	Color Space Representations	10
2.2.1	Linear	10
2.2.2	sRGB	11
2.2.3	CIELAB	12
3	IMAGE DESCRIPTORS	13
3.1	Statistical Moments	13
3.2	Scale Space Concepts	15
3.3	Gabor Descriptor	16
3.3.1	Gabor Kernel	17
3.3.2	Gabor Filter Bank	18
3.3.3	Distance Metric	19
3.4	Histograms of Oriented Gradients	19
3.4.1	Computing the Descriptor	19
3.4.2	Descriptor Size	20
4	MATERIAL MODELS	21
4.1	Radiometry	21
4.2	The Rendering Equation	22
4.3	BRDFs	22
4.3.1	The Lambert BRDF	23
4.3.2	The Phong BRDF	24
4.4	Textures	24
4.4.1	Displacement Maps	25
4.4.2	Bump Maps	25
4.4.3	Normal Maps	25
5	PROCEDURAL TEXTURES AND NOISE	27
5.1	Procedural Textures	27
5.2	Noise	28
5.2.1	Perlin Noise	28
5.2.2	Turbulence	29
III	PARAMETER RETRIEVAL FOR PROCEDURAL SURFACE GEOMETRY	31
6	RELATED WORK	33
7	PRELIMINARIES	35

7.1	Assumptions	35
7.1.1	Lighting	35
7.1.2	Materials	36
7.2	Choice of Descriptor	37
7.3	Space for Structural Comparison	38
7.3.1	Color Space	39
7.3.2	Remapping of values	39
7.4	Texture Models	41
7.4.1	Light Parameter	41
7.4.2	Amplitude Parameter	42
7.5	Target Images	42
8	FINDING STRUCTURAL MATCHES	45
8.1	Renderings	45
8.2	Descriptors and Caches	45
9	FINDING SHADING PARAMETERS	47
9.1	Using the Descriptor	47
9.2	Color Spaces and Similarity	50
9.2.1	Fitting in Linear	50
9.2.2	Fitting in sRGB	52
9.3	Color	53
10	IMPLEMENTATION DETAILS	55
10.1	Caches and Disk Space	55
10.2	Retrieval and Fitting Performance	56
10.2.1	Computing the Descriptor	56
10.2.2	Parameter Retrieval	56
11	DISCUSSION OF RESULTS	59
11.1	Lighting	62
11.2	Limitations	63
11.2.1	Abilities of the Model	63
11.2.2	Size of the Parameter Space	63
11.2.3	Quality of Shading Parameters	64
IV	CONCLUSION	67
12	FUTURE WORK	69
12.1	More Complex BRDFs	69
12.2	Capturing Additional Information	69
12.3	Evaluation	70
12.4	Summary	70
	BIBLIOGRAPHY	71

LIST OF FIGURES

Figure 2.1	CRT gamma and gamma correction	11
Figure 3.1	Images with equal histograms, but large perceptual differences	14
Figure 3.2	Anscombe’s quartet	15
Figure 3.3	Scale space examples	16
Figure 3.4	Checkerboards with equal histograms	16
Figure 3.5	Standard deviations on different scales	17
Figure 3.6	Results of convolutions with Gabor kernels	18
Figure 4.1	A sphere rendered with different BRDFs	23
Figure 4.2	Normal map based on a height field	26
Figure 5.1	Classification of procedural textures	28
Figure 5.2	Perlin noise	29
Figure 5.3	Examples of turbulence function with increasing persistence	30
Figure 7.1	Examples for kernels in the filter bank	38
Figure 7.2	Examples for images in comparison space	41
Figure 7.3	Available procedural textures	42
Figure 7.4	Locations of possible light sources	43
Figure 7.5	Effect of the amplitude	43
Figure 8.1	Pipeline overview	45
Figure 8.2	Example renderings including relief and normal map	46
Figure 8.3	An input image and the respective match	46
Figure 9.1	Input image and matches in comparison space	47
Figure 9.2	Process for matching the shading parameters	51
Figure 9.3	Adjusting a linear rendering	51
Figure 9.4	Process for matching the shading parameters via an sRGB match	52
Figure 9.5	Comparison of shading parameters found in linear and sRGB	53
Figure 9.6	Example of a match that considers surface color	54
Figure 9.7	Examples for matches created using sRGB and CIELAB	54
Figure 10.1	Input and matches for performance measurements	57
Figure 11.1	Results with varying lighting	59
Figure 11.2	Surface plots	60
Figure 11.3	Stretching and rotated inputs	62
Figure 11.4	Lighting matches	62

LIST OF TABLES

Table 1	Cache sizes	56
Table 2	Performance measurements	57
Table 3	Results for grayscale input images	61
Table 4	Results for colored input images	66

Part I

INTRODUCTION

MOTIVATION

The definition of realism that I like the most is the one I first heard from my colleague, then at Pixar, Alvy Ray Smith: he claimed photorealism was roughly equivalent to visual complexity. Two factors underlie visual complexity, diversity in the types of primitives and their shear number.

— Pat Hanrahan [Ebe03]

In recent years, content production for both the games and the movie industry has grown increasingly more complex, to achieve greater visual quality. In the context of movie production and games this directly relates to the amount of assets being used as characters, props and environments. Entirely new algorithms have to be developed to cope with the geometrical complexity of modern scenes, for example regarding visibility culling [RFS] or the simplification of assets that are built from thousands of elements [Coo+07]. Additionally, synthetic images have to integrate into photography and match the visual quality by being photorealistic or at least *photosurrealistic* [AGBoo]. Examples are *visual effects* (VFX) for motion pictures, fully computer-generated photorealistic commercial spots for the car industry, architecture visualizations or realistic games. In all these fields, automating tasks can help to unburden artists and to meet tight deadlines.

With the advent of physically based shading for visual effects and computer generated feature animation [McA+12; HVS13], a lot of the work of look development has been transferred into more technical domains of the process like shader development. Before, artists hand-crafted the appearance of surfaces in specific lighting conditions and great care had to be taken both when combining different assets in a scene as well as when putting these assets into entirely different lighting setups. Now, artists can design appearances that look correct in a multitude of lighting conditions. They can also freely combine different assets as they all obey the laws of physics and look accordingly when brought together into a common scene. While this development shows, on the one hand, that there is a big interest in simplifying the work even if that requires mathematically more challenging processes, it has also taken some of the artistic parts of a production into the realms of maths and physics. This allows software to potentially automate even more aspects of the work that will come together seamlessly in the end without the need for human intervention, as they are bound by the laws of nature first and foremost instead of aesthetics.

Even though assets work correctly when combined and across multiple light setups now, designing these assets is still a lot of work. The two main responsibilities to consider in look development are designing varying surface detail and finding the shading parameters that result in the desired appearance.

Regarding the first task, one has two main options for creating surface detail via texturing: using images or *procedural textures*. To combine their advantages and disadvantages based on specific needs, animation studios have come up with hybrid processes and new technologies. For the production of the movie “Meet the Robinsons”, Disney has started to bake procedural textures into images, for example [BLo8]. Pixar developed a system for synthesizing textures from small, painted examples for the movie “Tangled” [Eis+10] and Weta Digital simulated sandstorms and flowing water to accurately depict the weathering of buildings and streets in their textures for the movie “Maze Runner: The Scorch Trials” [Sey15]. These last examples illustrate the importance of automating texturing by making use of procedural processes, as they ease productions that require many assets to be textured.

The goal of look development often is to match real world objects. Photos of these objects are commonly used as reference, but they are also a valuable input for an automatic process that finds suitable parameters for procedural materials. In this work an abstraction of the visual features in the input is used to estimate the underlying material properties. The ultimate goal will be to match the material depicted in the reference as well as possible with minimal required inputs and interactive performance. This allows to capture input images on set and quickly turn the data into assets. Having a system that enables such uncomplicated matches would be especially useful in the growing fields of *virtual production*, *virtual reality* and *augmented reality* as they will increase the demand for virtual assets further.

To summarize, automating tasks in look development has great potential to significantly save time and production costs. The key components for a suitable system are the retrieval of information about varying surface detail and shading parameters. This work proposes methods for finding this data. We concentrate on procedural textures, which will be used to generate displacement maps for bumpy surfaces to approximate the geometrical structure presented in input images. Our contribution builds on work by Gieseke et al. [Gie+14], who employed the same approach for color textures which can be used as maps for the albedo of surfaces, for instance. The albedo will be non-varying in our work, but will be retrieved based on the input images to match a depicted material as closely as possible regarding overall structure and color.

1.1 PROBLEM STATEMENT

We solve the problem of creating a procedural material based on an example photography. To this end, the parameters of a procedural displacement map, which approximates the geometric structure of the given example, need to be retrieved. Finding appropriate parameters will be achieved by using image descriptors that abstract the visual features of the input. Comparing the features of the example to those of synthetic images will allow the determination of the best matches.

Additionally, we will analyze the example's reflection properties to create a better overall match. To this end, the shading parameters that are used in a particular shading model need to be retrieved.

With this work we contribute to the area of automatic look development and implement a system with minimal inputs that reproduces surface appearance. We deliberately restrict the inputs to a single image, which is captured under unknown lighting conditions to make the system as flexible as possible. Additionally, performance that allows interactivity is the goal.

1.2 OUTLINE

The following part of the thesis will provide the fundamentals of the processes described later on. Images and image descriptors will be introduced in Chapter 2 and Chapter 3, respectively. Afterwards, rendering related topics such as material models are discussed in Chapter 4 and procedural textures in Chapter 5.

The next part examines related work in Chapter 6 and continues with a description of our method in Chapters 7, 8, 9 and 10 before closing with a discussion of the results in Chapter 11.

We conclude the thesis in Chapter 12, where we will discuss future work.

Part II

THEORETICAL FOUNDATIONS

We commonly refer to the result of photography as images. For our method we assume images of surfaces as inputs, we will extract information from them and create new ones that show a procedural material. Thus, images are of special importance and require a formal definition. How image values are represented is crucial in a lot of applications and we will briefly introduce color spaces for that reason.

2.1 DIGITAL IMAGES

There is a multitude of possible sources of digital images. Images are used in medical imaging and can encode density information about tissue, for example. They can also store the results from processes like electron microscopy. For our work we consider images that store information about electromagnetic waves that are perceivable by the human visual system.

Definition 2.1. *A grayscale image $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}$ is a two-dimensional, continuous, differentiable function defined over a rectangular domain Ω and maps to a scalar.*

The image domain is usually given as $\Omega = [0, w) \times [0, h)$, where w and h are the width and height respectively. It is thus a rectangle with possible positions (x, y) given in spatial coordinates. We will refer to $I(x, y)$ as the *intensity value*.

While grayscale images are useful for a lot of applications, usually color images are of greater interest, as they are able to replicate our own perception much more closely.

Definition 2.2. *A color image $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ is a two-dimensional, continuous, differentiable function defined over a rectangular domain Ω and maps to a triplet.*

Definition 2.2 extends Definition 2.1 with the notion of *color channels*. A channel $c \in \{1, 2, 3\}$ captures only a subset of the visible light spectrum. We will refer to $I(x, y, c)$ as the intensity value in a particular channel c . $I(x, y)$ shall refer to a triplet representing the intensity values of all color channels at position (x, y) . Color channels usually refer to red, green and blue channels, in this particular order. For that reason, color images are often referred to as RGB images. The prop-

erty in Eq. (2.1) commonly holds for their intensity values, but for some applications $I(x, y, c)$ exceeds the upper limit:

$$0 \leq I(x, y, c) \leq 1 \quad (2.1)$$

The image values usually represent continuous physical properties. To convert this data into a digital representation, two steps are necessary. *Quantization* replaces the co-domain of continuous intensity values by a set of discrete values in order to represent them efficiently in binary form. *Sampling* on the other hand replaces the continuous spatial domain by a set of distinct positions at which the image value is given. We will refer to these point samples as *pixels* [Smi95].

Definition 2.3. *A digital image is an image whose spatial domain as well as its value domain are finite and discrete.*

In the context of image processing the rate of change of the intensity values are often of special interest. The first-order derivatives of a digital image I can be approximated using finite differences. Together they make up the *gradient* of an image:

$$\nabla I = \text{grad}(I) = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix} \quad (2.2)$$

For a univariate function $f(x)$ the central difference approximation is defined as follows:

$$\frac{\partial f}{\partial x} = \frac{f(x+h) - f(x-h)}{2h} \quad (2.3)$$

For a multivariate function the derivatives of the respective dimensions can be approximated respectively while fixing the other dimensions. Note that $h = 1$ in the case of a digital image. More information about digital images can be found in Gonzales and Woods [GW07].

2.2 COLOR SPACE REPRESENTATIONS

The pixel values of an image can be transformed, so that they are represented more conveniently for specific applications. The image stays intrinsically the same, but the way it is represented can change drastically with important side-effects. This section is about these transformations and why they are done, as we will work with images in different spaces in following chapters.

2.2.1 Linear

We are concerned with images that result from measuring electromagnetic waves that are visible to the human observer. Important

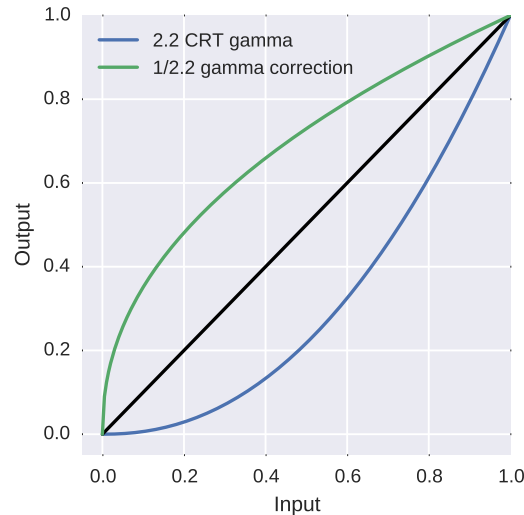


Figure 2.1 – The interconnection between CRT gamma and gamma correction to recreate a linear relationship. To compensate for the CRT gamma, a gamma correction has to be applied.

properties of these waves should therefore be at the core of our computations. Light transport is inherently linear and therefore a linear relationship between observed light and intensity values should exist. That means that the combined contribution of two light sources are a sum¹. It also means that, if we double the amount of light that falls onto our digital sensor, the response as represented in pixel values should be twice as high. While these requirements pose some difficulties for camera manufacturers, it is straightforward to compute with linear values. They are not suitable for display devices, though.

2.2.2 sRGB

Different color spaces come into play because the human visual system does not respond in a linear manner to light [Ste61]. *Cathode ray tubes* (CRTs) also do not emit light in a linear relationship to input voltages. The cause for this behavior are electrostatic effects in the electron gun [Rob05]. While modern displays don't necessarily have this problem, they are usually built to replicate a CRT's behavior for compatibility and historic reasons. These screens apply a 2.2 display gamma which has to be compensated for beforehand to maintain a linear relationship between code values and emitted light. Fig. 2.1 visualizes the problem.

The sRGB [ISO04] color space allows displaying images on these kind of screens directly. As opposed to just being a pure inverse gamma for the 2.2 CRT gamma, it has a linear foot that improves the distribution of code values for dark values. Often, an image that

¹ We ignore interference effects here as they are not important for our argumentation.

has the simple inverse gamma correction applied is referred to as being in sRGB space. While technically incorrect, the naming illustrates that the image can be shown on a screen directly. For our work only the inverse screen gamma is applied when we talk about sRGB space.

2.2.3 CIELAB

CIELAB [ISO08], or Lab, is an alternative to the RGB representation. It is three-dimensional as well, with an L dimension that encodes luminance (lightness) and two dimensions a and b, which encode colors on one axis that extends from green to red and one that extends from blue to yellow. The goal of CIELAB is to provide a space in which the euclidean distance between colors is proportional to their perceived differences [Fai13].

To approximate the surface structure that is visible in a given input image, a suitable set of parameters for a procedural height field needs to be found. We will thus have to be able to compare the input image to a synthetic rendering. This requires an abstraction of the visible features, as simply comparing the image values at all pixel positions is not invariant to small changes, and it can not mimic the way human perception works. *Image Descriptors* describe an image, or a region thereof, and abstract the features formerly represented in the intensity values and their ordering into a different, usually more compact expression.

To describe an image, one can differentiate *external characteristics* and *internal characteristics* of a region. External characteristics relate to the boundary of a region, internal characteristics to the pixels inside the specified area. What kind of descriptor to choose depends heavily on its application. For instance, if one is primarily interested in shapes within the image, segmenting the image domain and using a descriptor that considers merely the boundaries of the segments might be suitable. Examples are *Chain Codes* [Fre61; GW07] and *Minimum-Perimeter Polygons* [KR04; GW07].

For the purpose of describing visible surface structure, we will concentrate on internal characteristics. We are interested in the structural differences of images, which are not guaranteed to contain any characteristic shapes. The interested reader is referred to Gonzales and Woods [GW07] for an overview of possible descriptors of external characteristics. We will drop the distinction between internal and external and always refer to the former from now on.

For the task at hand it is of great importance to have a suitable descriptive representation of image features, which is expressive enough to find perceptually similar matches, but compact enough to allow interactive performance. We will discuss several possibilities therefore, so that we can properly justify our choice for our implementation in Chapter 7.

3.1 STATISTICAL MOMENTS

Using *statistical moments* is one of the simplest approaches for describing an image. It computes statistical measures of image value distributions, or the respective histograms, and the resulting *moments* are the entries of a descriptor vector. Differences between these vectors then express differences in the features of the described images.

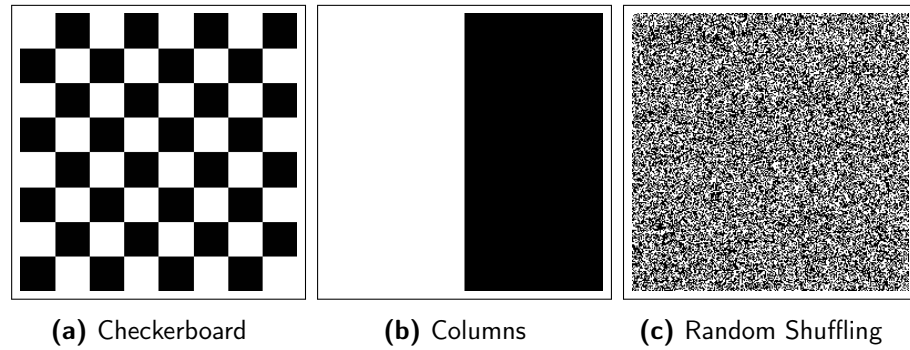


Figure 3.1 – Images with equal histograms, but large perceptual differences.

The general formula for computing the statistical moment of order k about the mean μ for an image with N pixels whose intensities are denoted as f_i is given as

$$\mu_k := \frac{1}{N} \sum_{i=1}^N (f_i - \mu)^k. \quad (3.1)$$

Important examples of these central moments include the *variance* μ_2 , the *skewness* μ_3 and the *kurtosis* μ_4 . While the variance is a measure of fluctuations around the mean and of intensity contrast, the skewness denotes the level of asymmetry in the histogram. The kurtosis measures *flatness* of the intensity distribution. Subtracting 3 from the kurtosis is often done for convenience, as it results in $\mu_4 = 0$ for a Gaussian distribution [DeC97]. We thus propose an exemplary descriptor vector $\{\mu, \mu_2, \mu_3, \mu_4\}$ for this theoretical discussion. The proposed description gives a first impression regarding the images it represents, but it has its limitations for image discrimination. Consider the fact that all of the aforementioned measures can be computed from a histogram [GW07]. It follows that the actual pixel position of each intensity value is irrelevant for the resulting descriptor vector. See Figure 3.1 for an example of images with identical histograms but very different structure. Since the descriptor vector is the same for each of the shown images, we can not distinguish the images based solely on their statistical description. Furthermore, the statistical measures can be equal for images with very different histograms as pointed out by Anscombe [Ans73]. Figure 3.2 shows his classic four data sets, which differ vastly but produce almost the same means μ_x and μ_y and almost the same variances σ_x and σ_y .

To alleviate the problems of missing spatial context, the neighborhood of the pixels need to be considered in the descriptor. Using scale space is a possible way to do so.

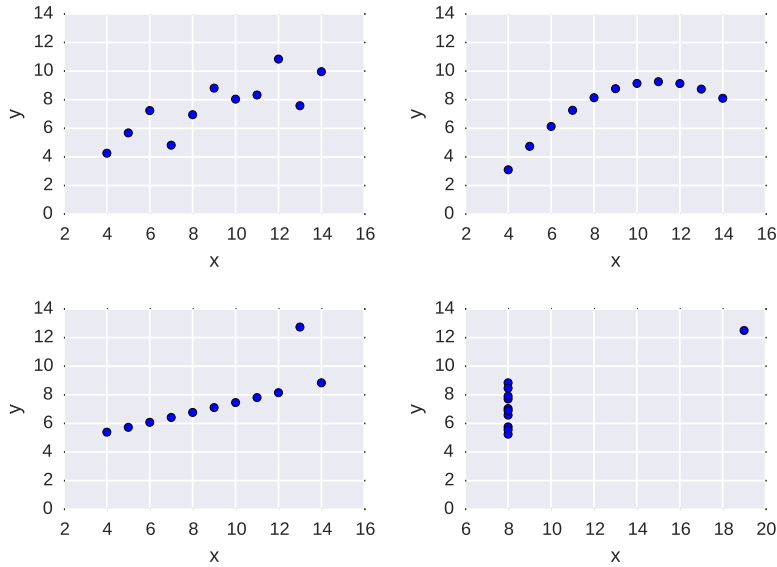


Figure 3.2 – Anscombe's quartet. Although the sample distributions differ, their means and standard deviations are almost the same.

3.2 SCALE SPACE CONCEPTS

Scale space concepts are important in a wide range of computer graphics applications. We will introduce scale space here for image description only and discard other possible usages. The interested reader is referred to [Lin13], for example.

The idea of scale space is that features in the image are only visible on certain scales, which is useful for a thorough image description and differentiation.

Eq. (3.2) shows that the space is parametrized using a single parameter σ , which is the standard deviation for a Gaussian kernel g_σ .

$$L(x, y, \sigma) = I(x, y) * g_\sigma(x, y) \quad (3.2)$$

Computing the convolution of the original image with this kernel results in the image at a certain scale. Convoluting with a Gaussian kernel is closely related to linear diffusion and both give equivalent results [Koe84]. Examples can be seen in Fig. 3.3.

Note how the prominent features in the image, edges and corners, start to dissolve and are removed entirely for a standard deviation of $\sigma = 32$. In Fig. 3.4 there are five different images showing a checkerboard with different frequencies. First order statistics are unable to differentiate the images as explained before since the histograms of the images are identical.

If we define a set of kernels, a filter bank, consisting of Gaussian kernels of varying standard deviations to represent a set of scales, convolve the images and look at the first order statistics of the results, we are able to differentiate between the checkerboards. This follows

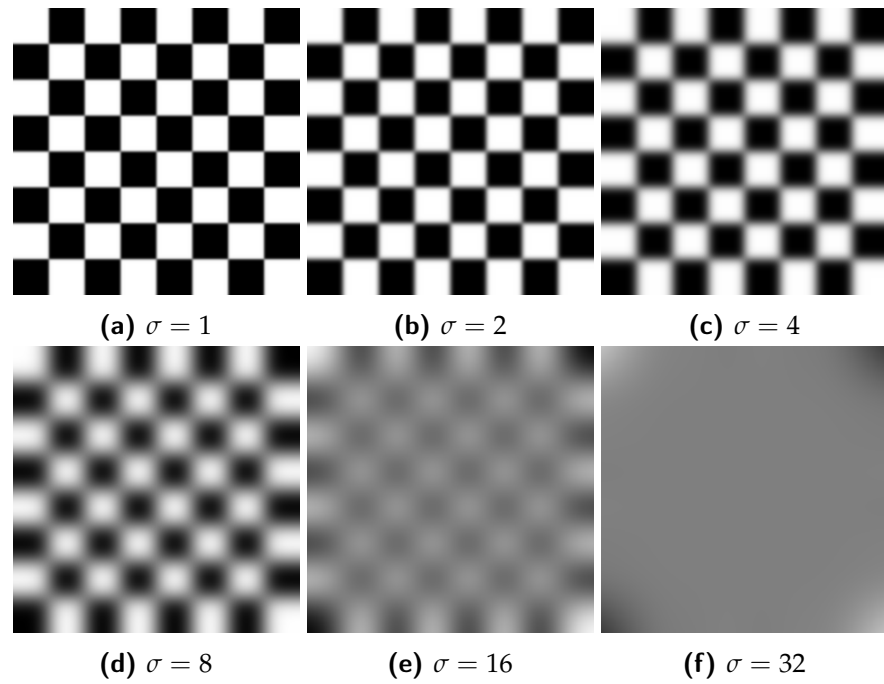


Figure 3.3 – Example scales and the respective standard deviations σ according to the definition in Eq. (3.2).

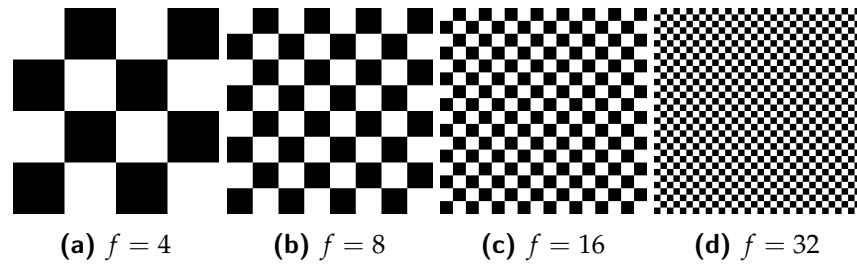


Figure 3.4 – Checkerboards with equal histograms but different frequencies f .

from Fig. 3.3 which shows how image features vanish for different kernel sizes. The visibility of details has a big influence on the first order statistics of the image on a particular scale. Fig. 3.5 shows that feature vectors containing the standard deviation after convolving the images with kernels of different sizes differ enough to discriminate the images. This image descriptor includes neighborhood information to address some of the problems of simple first order statistics.

3.3 GABOR DESCRIPTOR

As seen before, convolving images with a set of kernels can help to differentiate them. The reason is that the first order statistics of the results may exhibit more differences than those of the images themselves do. The set of kernels to be used has great influence on the

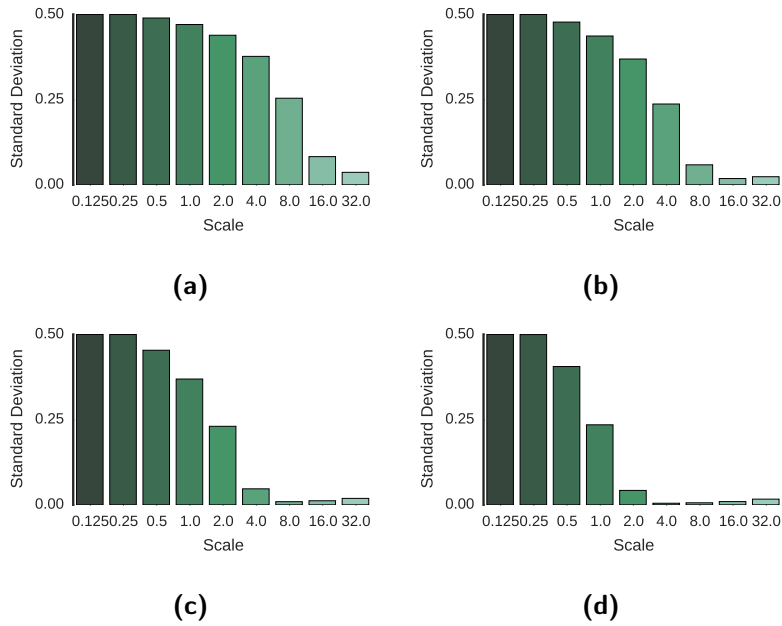


Figure 3.5 – Standard deviations of the inputs in Fig. 3.4 on different scales σ .

quality of the results and different kernels have different advantages. Since the Gaussian kernels used before are rotationally invariant, they can not discriminate images that differ only in the rotation of the image contents. They also can not detect edges. The following section describes a kernel that can alleviate these problems.

3.3.1 Gabor Kernel

A definition of the Gabor function in two dimensions can be seen in Eq. (3.3):

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) := \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cdot \cos\left(2\pi \frac{x'}{\lambda} + \psi\right) \quad (3.3)$$

Its definition varies slightly in the literature [PD05; MM96], but it is always defined as the product of a sinusoid and a Gaussian function. Rotating the kernel is achieved by transforming the coordinates following the scheme in Eq. (3.4) and Eq. (3.5):

$$x' = x \cdot \cos(\theta) + y \cdot \sin(\theta) \quad (3.4)$$

$$y' = -x \cdot \sin(\theta) + y \cdot \cos(\theta) \quad (3.5)$$

The parameters for $g(x, y; \lambda, \theta, \psi, \sigma, \gamma)$ shape the kernel and determine the frequency λ of the sinusoid, the orientation θ of the kernel, the phase offset ψ of the sinusoid as well as the standard deviation σ and the aspect ratio γ of the Gaussian.

The concept of scale space is directly transferable to the Gabor kernel. The standard deviation σ has the same effect on the kernel size

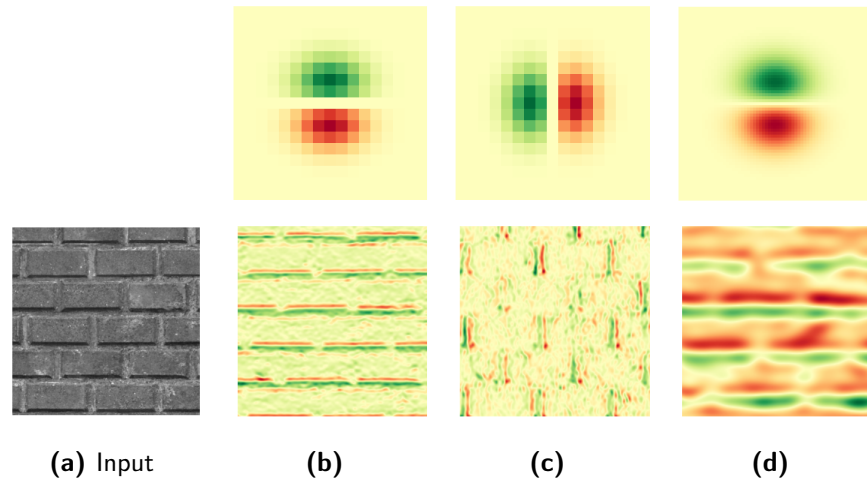


Figure 3.6 – Three kernels (b-d, top) and the resulting images (b-d, bottom) after convolution. Both are color coded so that negative values are shown in red, positive values in green.

as discussed before and a filter bank with varying standard deviation can be regarded as defining a scale space. Fig. 3.6 shows the shape of example kernels and the results of convolving an image with them.

The shapes of the kernels determine the kinds of local structures in the image that the kernel responds to. For the first two examples in 3.6b and 3.6c, either horizontal or vertical lines are detected. The last example in 3.6d shows that a kernel with a bigger σ detects larger local elements in the image. A feature vector that consists of first order statistics of these responses can be used to discriminate images with differently rotated structures as well as structures that have been scaled.

Gabor kernels are popular for a variety of reasons. Firstly, dominating directions can be detected and edges or stripes in the image lead to different responses based on the standard deviation σ and frequency λ used. Another reason is the similarity between the gabor functions and the way the visual system of humans work, making Gabor functions particularly useful for pattern recognition. Cells in the visual cortex respond to structures like Gabor functions, leading to the conclusion that Gabor functions are able to detect the same kind of patterns as the visual systems of different species of mammal [Pou11]. The interested reader is kindly referred to available research, such as the work by Marçelja [Mar80].

3.3.2 Gabor Filter Bank

Deciding which kernels to include in a filter bank determines which structures can be detected and thus the quality of the image descriptor. The size of the filter bank is also directly proportional to the descriptor's size.

Manjunath and Ma [MM96] have designed a filter bank that reduces redundancy. In this context, redundancy refers to overlaps regarding the local structures that kernels are able to detect. Ideally the kernel responses would cover the entire frequency spectrum of an image without overlap. Their filter bank is particularly interesting in the context of image descriptors as they have shown that their set of kernels can be used successfully to retrieve images with similar features as an input image.

Additionally, Gieseke et al. [Gie+14] use a similar set of kernels to retrieve the parameters for procedural textures. Their work is directly related to ours.

3.3.3 Distance Metric

To determine the distance between two descriptors, a distance measure based on the entries of the descriptors is needed. Gieseke et al. [Gie+14] propose to model the histograms of the filter responses as Gaussian distributions so that the respective mean μ and standard deviation σ can be compactly stored in the descriptor vector. The distance between two distributions can be computed using the Wasserstein metric [GS+84]. It is also called “Earth Mover’s Distance” (EMD) as it reflects the cost of redistributing mass in order to match another distribution. The EMD is commonly used in the context of image retrieval [RTG00] and is a popular metric for image similarity [Chao2]. Gieseke et al. [Gie+14] argue that the distance between their descriptors, based on the EMD, can be simplified to an L_2 distance in this particular case.

3.4 HISTOGRAMS OF ORIENTED GRADIENTS

Another descriptor that is concerned with orientations in an image is called “Histograms of Oriented Gradients” (HOG) [DT05]. It is used mostly to detect humans in images, but can be used successfully in the context of image classification [Lin+11] as well. A variation of HOG is used for sketch based image retrieval [HBC10].

3.4.1 Computing the Descriptor

To compute the HOG descriptor, the image window is divided into *cells*. In the original work by Dalal and Triggs [DT05], an image window of 64×128 pixels is used, which is divided into cells of 8×8 pixels each. Gradients or edge directions are then computed for the pixels in a cell. Afterwards, the directions are combined into a histogram with nine bins for orientations in $[0^\circ, 180^\circ]$. A Gaussian kernel can be used to weight the directions based on the respective pixel’s position in the cell. To make the descriptor invariant to local illu-

mination changes and contrast differences, normalization has to be applied. Different options for normalization exist and the default descriptor described by Dalal and Triggs [DT05] combines four cells into a 16×16 *block* and uses this larger region for normalization. Blocks overlap by 50%, so that each cell is covered four times and contributes its 9-dimensional representation four times to the resulting descriptor.

3.4.2 *Descriptor Size*

While the size of the convolution based descriptors discussed previously depends only on the size of the filter bank that is used, the descriptor size for HOG scales according to the size of the image region, of the cells, and of the blocks. In the work by Dalal and Triggs [DT05] the default window size of 64×128 is composed of $7 \times 15 = 105$ blocks. With four cells per block and nine bins in the histograms, the resulting feature vector for this relatively small image window has 3780 entries.

Our goal is to not only match surface structure but also reflection properties to obtain a good procedural approximation of the material depicted in the input image. Nature offers a wide range of such materials and different surfaces. By perceiving their properties through our visual system, our ability to differentiate between objects and to navigate the world is enhanced. In fact, “we acquire more information through vision than through all of the other senses combined” [War12]. The way objects interact with light is of great interest for that reason, as it is the main source of information about the environment. It is thus easy to understand why the simulation of light interactions is a major topic in computer graphics and a lot of research has been guided towards understanding the involved phenomena.

The main characteristics of a material are its general properties regarding the scattering of light as well as modulations that may vary certain aspects of light interactions across the surface. Part of the second aspect is geometrical detail like bumps and cracks that change how a surface reacts to light. In order to match a surface patch given in an input image, we will look at both and discuss necessary fundamentals.

4.1 RADIOMETRY

It is important to briefly introduce the radiometric unit *radiance* here, as it is essential for most computations regarding the interaction of light and surfaces. For a more thorough discussion of radiometry, the interested reader is referred to Pharr and Humphreys [PH04] or Dutré et al. [DBB06] upon whose work this section is loosely based. Radiance is a central unit used for computing light transport as it is able to capture the notion of “amount of light” in a convenient way. Its definition can be seen in Eq. (4.1):

$$L = \frac{\partial^2 \Phi}{\partial \omega \partial A^\perp} = \frac{\partial^2 \Phi}{\partial \omega \partial A \cos \theta} \quad (4.1)$$

Thus, radiance can be described as “flux per unit projected area per unit solid angle” [DBB06]. The flux Φ measures the energy that interacts with the surface area A . Defining the radiance with regard to the solid angle ω and the projected area A^\perp creates the properties of radiance that justify its wide usage.

Radiance is of special interest when it comes to simulating light in the context of computer graphics for two main reasons. Firstly, sensors and the human visual system respond proportionally to the

received radiance. The same can not be said about other radiometric quantities such as flux or irradiance ($E = \frac{\partial\Phi}{\partial A}$). Secondly radiance does not change along a straight path or a ray:

$$L(x \rightarrow y) = L(y \leftarrow x) \quad (4.2)$$

It thus captures the “appearance” of surfaces independent of the distance of the observer to the object.

4.2 THE RENDERING EQUATION

With the definition of radiance, we can now look at the rendering equation [Kaj86], shown in Eq. (4.3), and understand how light transfer can be modeled in a formal mathematical way.

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega_o) L_i(\mathbf{x}, \omega_i) \cos \theta d\omega_i \quad (4.3)$$

The subscripts i and o denote incoming and outgoing directions, where outgoing emission always refers to the amount of light emitted towards the observer from the point \mathbf{x} . The sum on the right-hand side consists of two parts, L_e and an integral over the hemisphere Ω^+ defined by the surface normal at point \mathbf{x} . L_e refers to the amount of light that is directly emitted at the surface. An example of an emitting surface would be the glowing coil in a light bulb.

The integral on the other hand refers to all the light that is transported to \mathbf{x} from above the surface and is then reflected towards the observer. While L_i refers to the amount of light coming from a certain direction within the hemisphere, the $\cos \theta$ term is a geometric term that makes up for the fact that a volume of photons is “spread” onto a differently sized area A for different angles θ measured towards the normal of the surface. Finally, f_r is the BRDF, the bidirectional reflectance distribution function, which is used to compute how the incoming light from a direction ω_i is reflected into different outgoing directions.

BRDFs are of utmost importance for the modeling of surfaces. The next section will give a brief introduction of common properties and will introduce two simple but widely used BRDFs.

4.3 BRDFS

BRDFs are meant to compute how much of the incoming light from direction ω_i is reflected into an outgoing direction ω_o at point \mathbf{x} . Different sources and representations of BRDFs exist [PHo4]. It is possible to create a table of values for possible inputs and simply look up or interpolate the data that was stored. While this is useful for measured BRDFs, this method also takes up a lot of disk space and it is costly to generate the table as it is usually done by measuring

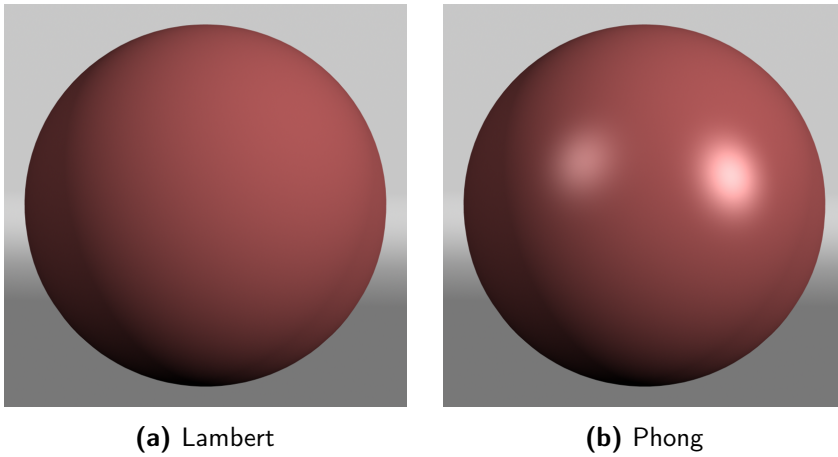


Figure 4.1 – A sphere rendered with different BRDFs.

actual surfaces. Furthermore, it is difficult to change the data if an artist wants to change the appearance of an object. For that reason one usually uses algorithmic BRDF models that use a few parameters which control the surface’s appearance and can be easily modified. We will focus on mathematical models here, as it is the objective of our work to avoid costly measurements.

One important property of BRDFs is Helmholtz reciprocity:

$$f_r(\omega_i, \mathbf{x}, \omega_o) = f_r(\omega_o, \mathbf{x}, \omega_i) \quad (4.4)$$

It means, that one can swap the direction of incoming and outgoing light and get the same result as light paths should be reversible. For measurements, it also means that one can swap the positions of the light source and the measuring device.

Physical BRDFs should also be energy conserving. The simple idea behind this principle is that a surface can not reflect more light than it received. One can express this idea formally as in Eq. (4.5) and derive important constants from it that make sure that surfaces behave physically plausible:

$$\int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega_o) \cos \theta \, d\omega_i \leq 1 \quad (4.5)$$

We will look at two specific BRDFs now.

4.3.1 The Lambert BRDF

The Lambertian model is one of the simplest BRDFs and its main characteristic is that it scatters incoming light equally into all outgoing directions. Thus, it is a simple constant as shown in Eq. (4.6).

$$f_r(\omega_i, \mathbf{x}, \omega_o) = \frac{\rho_d}{\pi} \quad (4.6)$$

The $\frac{1}{\pi}$ term is the result of the principle of energy conservation and can be derived directly from Eq. (4.5). The albedo or diffuse reflectance ρ_d is a constant that describes how much of the incoming light is reflected off of the surface independent of direction. Intuitively, one can regard it as the parameter to tune when one wants to represent the difference between a darkly tinted piece of paper and a purely white one. Typically a texture is used for the albedo to control varying surface detail.

The surfaces that this idealized model can describe are purely diffuse and do not show any specular highlights. It is merely an approximation to real world surfaces, but the Lambertian model is widely used in computer graphics applications. A rendering using it can be seen in Fig. 4.1a. A more physically plausible model for diffuse surfaces is the Oren-Nayar model [ON94], which provides a parameter for the roughness of the surface.

4.3.2 The Phong BRDF

The Phong BRDF is a simple phenomenological model that is able to compute specular highlights. The highlight is modeled by $\cos(\theta_s)^p$, where θ_s is the angle between the viewing vector V and the vector for ideal reflection R [Pho75]. The parameter p determines the roughness of the surface, i. e. the spread of the cone into which incoming light is reflected. The cosine can be computed as a dot product between the two vectors if they are of unit length which makes the Phong model easy to implement and fast to evaluate. Often, the Phong model is used as

$$f_{\text{phong}} = k_d \cdot \cos(\theta) + k_s \cdot \cos(\theta_s)^p, \quad (4.7)$$

in a context that evaluates local illumination based on direct light sources. While this is not physically accurate, it is a widely used model for reflective surfaces. Here, k_d and k_s are coefficients that weight the contribution of diffuse and specular lighting and can take on different values for red, green and blue. A rendering using the Phong model can be seen in Fig. 4.1b.

Some enhancements have been proposed among which is work that proposes a scaling factor to make Phong reflectance energy conserving [Lew94]. More realistic models are often based on microfacet theory like Cook-Torrance [CT81] or Ashikhmin-Shirley [AS00].

4.4 TEXTURES

Texture mapping [Cat74] is a standard way to add details to the surface of a digital asset. Various *maps* can be used to modulate various characteristics of the material across a surface. It is quite common

to employ diffuse and specular maps, for example. They are multipliers for the diffuse and specular response of a material as shown in Eq. (4.7). Now that physical models are used more and more in production, roughness maps have replaced specular maps in a lot of situations. They modulate the actual behavior of the BRDF instead of simply scaling its output which results in more realistic materials. Roughness maps are particularly useful for BRDFs that follow microfacet theory and modulate the normal distribution function used in these models.

Texture maps can also be used to add geometrical detail to a surface. The three main examples for feature film animation and visual effects are *displacement maps*, *bump maps* and *normal maps*.

4.4.1 Displacement Maps

Displacement maps define the distance by which a point on a surface is *displaced* in the direction of the surface normal. As such, they are used to actually change the surface which results in correct shadows, reflections and silhouettes. In raytracers displacement maps are somewhat costly as they require the renderer to finely tessellate the geometry so that individual patches can be displaced according to the image values. These are usually given in a grayscale image.

4.4.2 Bump Maps

Bump maps are inherently the same as displacement maps, but are treated differently by the renderer. Instead of changing the surface geometry, they simply cause a recomputation of the surface normals. This results in incorrect shadows etc., but can be used successfully to modulate a surface's appearance without too much computational overhead.

4.4.3 Normal Maps

Normal maps behave similar to bump maps in that they do not really change a surface's geometry. They are used to directly read the surface normal from the image, without having to compute new normals based on point offsets. They are given as images with three channels with intensity values $I(x, y, c) \in [-1, 1]$, or $I(x, y, c) \in [0, 1]$ for displaying them. The three channels are used to store a three-dimensional vector. When it comes to objects that move in space or might even be deformed, one has to be careful about the space in which the normals are stored in the image. Often, *tangent space* is used to represent the normal independently from a surface's transformation.

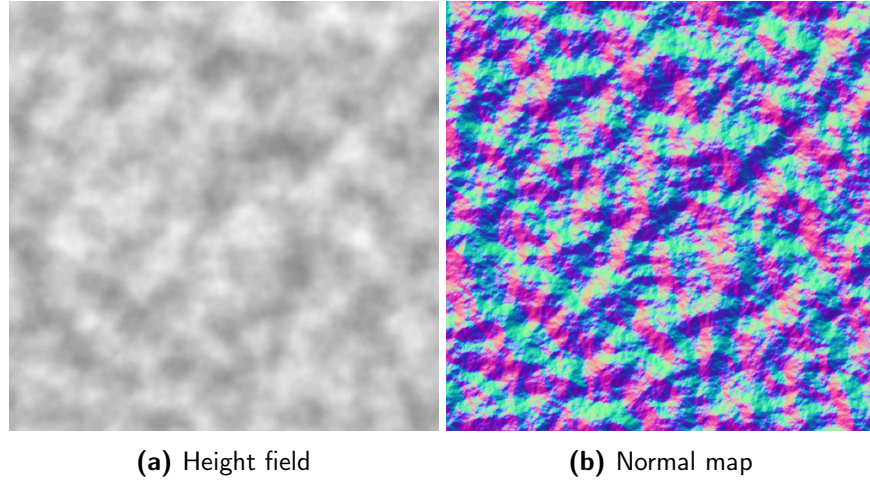


Figure 4.2 – Normal map based on a height field.

For displacement maps that are used as a height map on a simple plane, it is straight-forward to compute the resulting surface normals. Reducing the computational complexity to reading a normal from disk can have great benefits without affecting the image quality if the actual surface point in three-dimensional space is of no interest for the lighting calculations, which is commonly the case when using a directional light. In the following, we will briefly introduce how to compute the normals for a height field as this process will be of interest later on.

We have seen in Section 2.1 how to compute the derivatives $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ for a digital image. In the case of a displacement map that is used as a height map, the pixel values are the z-coordinates of the height field.

$$p := \frac{\partial z}{\partial x} = \frac{\partial I}{\partial x} \quad (4.8)$$

$$q := \frac{\partial z}{\partial y} = \frac{\partial I}{\partial y} \quad (4.9)$$

Thus, the tangents of a point on the displaced surface are given as $(1, 0, p)^\top$ and $(0, 1, q)^\top$. The normal can then be computed via a cross product.

$$\mathbf{n} = \begin{pmatrix} 1 \\ 0 \\ p \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ q \end{pmatrix} = \begin{pmatrix} -p \\ -q \\ 1 \end{pmatrix} \quad (4.10)$$

Additional normalization to obtain a normal of unit length is common and often needed as cosine terms are often evaluated as dot product between vectors of unit length.

In order to create synthetic surface detail to match an input image, a way to generate the required patterns is needed. In nature, color variations and geometric structures are the results of complex processes and contribute to the characteristic look of a material. As discussed in 4.4, textures can be used to add surface detail that resembles natural patterns. Varying colors can be mapped to the albedo ρ_d (see 4.3.1) and displacements can change the geometrical properties of the surface. Through this added complexity, photorealism in synthetic images is greatly enhanced. In this chapter we discuss methods of creating the necessary patterns.

5.1 PROCEDURAL TEXTURES

Two options regarding textures are images and procedural textures. Using images has the advantage of complete artistic freedom, but a few drawbacks, that are worth addressing. Firstly, images have a fixed resolution which can lead to filtering issues. MIP mapping can help with these problems [Wil83]. The size of images are of special importance as feature film vfx projects can end up using thousands of textures that need hundreds of gigabytes of disk space [Inc15]. Apart from disk space, images also consume a lot of memory and network bandwidth which can lead to performance issues. An example are situations in which missing image data leads to cache misses on the CPU [Eis+13]. Procedural textures, on the other hand, are described by an algorithm as opposed to data structures stored on disk. They have certain advantages as outlined by Lagae et al. [Lag+10c] and briefly summarized in the following.

Procedural textures can be compactly stored, as only a set of parameters need to be kept in memory to compute the textures. Being parametrized, they can usually represent a range of patterns instead of just one particular expression. They are evaluated on-the-fly on the CPU and can be filtered according to the distance to the camera, i. e. the size of the projected area in screen space [Cor07], but can also lead to aliasing effects if one is not careful [Ebe03]. Further advantages are the facts that procedural textures are usually non-repeating and have no visible seams. Especially with parallel computations in mind, it is important to note that procedural textures are often randomly accessible and can be evaluated in constant time. A problem of procedural textures is their lack of art directability. While a lot of phenomena in nature can be reproduced with a wide range of pro-

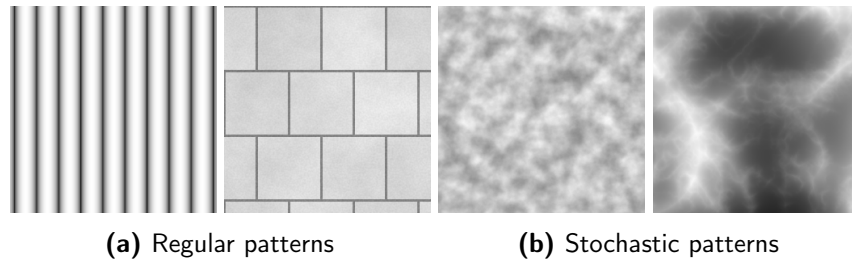


Figure 5.1 – Procedural textures classified according to the spectrum proposed by Lin et al. [Lin+06].

cedural textures, their usually non-intuitive parameter space makes it difficult for artists to achieve the desired look. Work by Gieseke et al. [Gie+14] addresses this problem.

Procedural textures can be classified according to where they are located on a *regular* versus *stochastic* scale [Lin+06] where *irregular* patterns are somewhere in between. Regular patterns are usually a collection of shapes placed regularly on the canvas. By keeping global regularities and adding local randomized offsets and patterns, we move along the scale towards entirely stochastic patterns such as the surface of rocks, for example. We commonly refer to these purely stochastic patterns as *noise*.

5.2 NOISE

Lagae et al. [Lag+10c] define noise as “the random number generator of computer graphics” and as a “random and unstructured pattern”. This pattern, the noise signal, is often described by its power spectrum which indicates the magnitude of the frequencies that contribute to the signal. Modeling a noise function can be done through modulating its power spectrum.

In their overview, Lagae et al. [Lag+10c] group noise functions into three categories: *lattice gradient noises*, *explicit noises* and *sparse convolution noises*. We will limit this discussion to the first, lattice gradient noises, and briefly examine the classic Perlin noise function and its application in the turbulence function [Per85] as an example.

5.2.1 Perlin Noise

Lattice noises have in common that they need a structured grid of points. This lattice is often referred to as the *integer lattice* because it places the samples at locations in texture space where the coordinates are integers (see Fig. 5.2a). Interpolating between the samples results in a continuous function. By recalling some fundamentals from signal processing, we can understand some implications regarding the resulting power spectrum. The Nyquist-Shannon sampling theorem

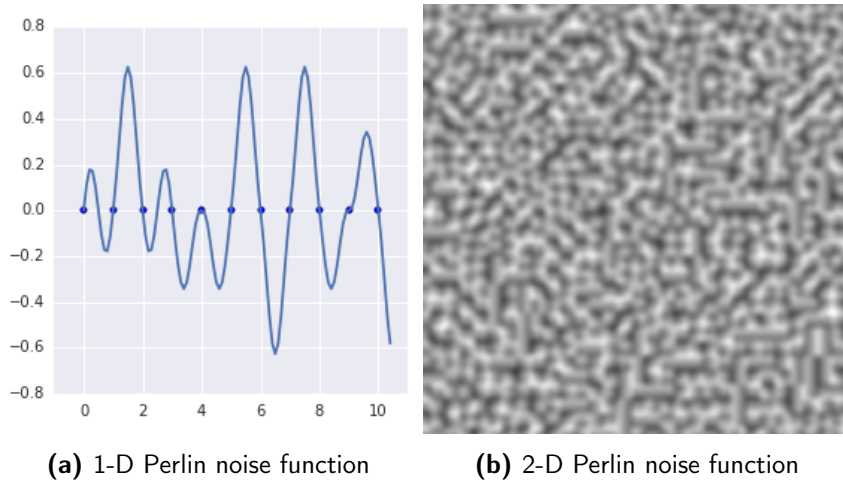


Figure 5.2 – Perlin noise, a gradient lattice noise.

states that to perfectly recreate a band-limited signal from samples, the sample rate has to be twice the highest frequency of the power spectrum (see [Jer77], for example). A signal that was constructed from the samples on the integer lattice should be band-limited accordingly and can be regarded as a low-pass filtered signal [Ebe03].

Simple value noise places scalars as samples on the grid and uses one of many possible interpolation methods to generate values in between. Gradient methods on the other hand place gradients on the lattice points and create the function from them. The function value for integer-valued coordinates is zero. Perlin’s noise function was the first implementation of a gradient noise [Ebe03]. Fig. 5.2 shows examples for one and two dimensions. Note that the frequency of the noise is easily controllable by scaling the coordinate for which to evaluate the function by a scaling factor f . It’s also possible to shift the noise by adding an offset o .

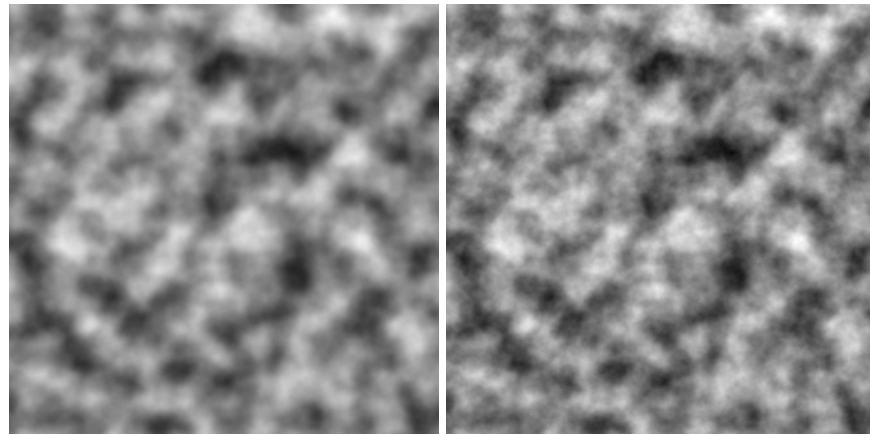
$$y = \text{noise}(f \cdot x + o) \quad (5.1)$$

5.2.2 Turbulence

It’s possible to modulate the power spectrum further by combining different Perlin noise functions into a *turbulence* function. Commonly, multiple noises of different frequencies f are summed up while weighting them inversely proportional to f , resulting in a “fractal” power spectrum [Ebe03]. Frequencies are commonly chosen to be powers of two as in Perlin’s [Per85] original implementation.

$$\text{turbulence}(x) := \sum_{i=0}^n \frac{\text{noise}(x \cdot 2^i)}{2^i} \quad (5.2)$$

The pixel resolution of the texture determines the upper limit n for frequencies to be used in Perlin’s original implementation. Replacing



(a) Low persistence values weight down high frequencies quickly

(b) Higher persistence values keep higher frequencies

Figure 5.3 – Examples of the turbulence function with increasing persistence.

the inverse weighting by a persistence parameter $p \in [0,1]$ allows further control over how quickly frequencies are decreasing in the power spectrum.

$$\text{turbulence}_p(x) := \sum_{i=0}^n p^i \cdot \text{noise}(x \cdot 2^i) \quad (5.3)$$

Fig. 5.3 shows examples of the turbulence function evaluated on a 2-D grid.

There are many more noise functions that can be used to generate procedural textures. Also, a multitude of procedural textures for regular or irregular patterns exist. Bricks, Voronoi cells and checkerboards are just a few examples. The interested reader is kindly referred to Lagae et al. [Lag+10a] and Ebert [Ebe03]. Now, with all necessary fundamentals covered, we can start looking at how to reproduce surface detail based on input images.

Part III

PARAMETER RETRIEVAL FOR PROCEDURAL
SURFACE GEOMETRY

RELATED WORK

Our goal is to retrieve the parameters for a procedural surface according to a given target. This section gives an overview of the current state of the art and analyzes where our work differs from existing publications.

Lin et al. [Lin+06] define a spectrum of different texture classes that spans from regular textures with recurring features to purely stochastic textures. Our system supports all texture classes and does not concentrate on one particular end of the spectrum as many other publications.

Texture synthesis is a family of data driven techniques that generate pixel data. Wei et al. [Wei+09] give an overview of the current state of the art in this field. These techniques are different from our approach. We use procedural textures and define our objective as a parameter retrieval problem.

Lagae et al. [Lag+10c] provide an overview of stochastic procedural textures. Analyzing input images allows for controlling parameters of stochastic noise functions to generate matching output. Examples for such methods are Gilet, Dischler, and Ghazanfarpour [GDG12], Galerne et al. [Gal+12] and Lagae et al. [Lag+10b]. Dischler and Ghazanfarpour [DG97] introduced a method for generating procedural “geometric” textures by decomposing stationary random signals. The resulting textures are compactly defined as a sum of elementary random functions. Varying geometric detail can be achieved using their method, but like the other methods mentioned before it is able to represent stochastic patterns only. We create surface detail without limitations regarding the texture class.

Gilet and Dischler [GD10] present a method to create stochastic volumetric and procedural detail. They allow a wide range of input photography and need user interaction to determine the areas of interest in the image. They also require the user to give a coarse representation of the geometry as well as an estimation of the lighting environment. Our work differs in that we do not need user input for geometry and lighting direction, but we require images that show a planar surface patch. We also do not impose restrictions regarding the class of procedural textures while Gilet and Dischler [GD10] generate stochastic textures only.

Wang, Snavely, and Marschner [WSM11] estimate surface structure visible in an input image that was captured under specific lighting conditions. Their inputs are shot under step-edge lighting and they estimate both a BRDF as well as a statistical model for the surface

geometry. The resulting surface structures are stochastic. We don't enforce any particular lighting conditions and allow non-stochastic surface detail.

Bourque and Dudek [BD04] retrieve the parameters needed for a procedural texture based on target images. Their work is not concerned with procedural geometric properties of surfaces like ours, but they also allow non-stochastic patterns in their system. They employ a distance metric to determine texture similarity and non-linear optimization to find the best match. Our technique differs in the method of retrieving the parameters. We use a sample-and-retrieve approach and achieve interactive results, while they report a cost of 12 minutes.

Gieseke et al. [Gie+14] extend the work by Bourque and Dudek by proposing an approach that allows interactive performance. By designing a sample-and-retrieve approach that samples the parameter space and precomputes caches that contain image descriptors, they allow retrieval times below one second. They propose a distance metric for texture similarity and impose no limitations regarding the class of the procedural patterns. Our work is built on theirs and extends it for procedural surface structure.

In order to retrieve suitable parameters for a procedural material, we will develop a set of techniques that find structural matches and approximate reflection properties. In this chapter we will have a look at all assumptions our method makes and discuss some specific decisions about descriptors and spaces afterwards.

We have seen before that image descriptors can be used to compare images. To obtain a good match for the geometric structure of the surface, the parameter space of procedural displacements needs to be explored. This will be done by generating the respective shaded images and computing the descriptors for these. We can then compare the resulting images to the input and find the closest matches. This approach is the same as employed by Gieseke et al. [Gie+14]. Our work extends their work by considering shading information. The parameter space is different for that reason and lighting directions need to be taken into account. We adopt the image size of 256×256 pixels for the input images as proposed by Gieseke et al. The reasoning for this size is that both high frequency detail as well as low frequency geometric structures can be represented sufficiently. The geometric structure is generated by procedural displacement maps and the parameter space of these should be limited to generate only detail that can be represented in the given image domain with the aforementioned resolution. The image descriptor to be used needs to be designed accordingly.

7.1 ASSUMPTIONS

To find the parameters for the displacement maps that replicate the input as closely as possible, we need to constrain the problem. In the following we will describe assumptions about the lighting and the materials.

7.1.1 *Lighting*

For the lighting in the input image we assume a single directional light with an intensity set to 1. All synthetic images are rendered using this setup. This assumption is a strong simplification of a realistic environment, but it is useful as it represents two important ideas. Firstly, the input images should contain some indications as to where light is coming from. Without it, in a purely diffuse environment, it would be very difficult to get a sense of the structure of the surface —

even for a human observer. Secondly, we limit these light sources to a single light that can be thought of as representing only the strongest direction in the image. Input images that show more complex lighting conditions can be used as inputs as well, but the results may be not as good as for simpler environments, as the synthetic renderings will never be able to replicate the input fully. We don't want to solve for the lighting environment here, though, but for the structure of the surface. Matching the lighting perfectly is not needed. Some information about the lighting can be obtained nonetheless as we will see.

Simplifying the lighting as described has some consequences for the computational costs as well. By using a single light source and ignoring indirect lighting, the computationally costly integral from the rendering equation is drastically simplified. Additionally, the factor that ensures that the Lambertian material is energy conserving can be dropped for simple punctual light sources¹ [Hof+10].

7.1.2 *Materials*

In nature a wide range of different materials exist as described in Chapter 4. Dealing with this variation of possible surfaces is difficult, because complex scattering behaviors require computationally costly calculations. For a lot of algorithms some simplifying assumptions about the surfaces are made. It is not uncommon in computer graphics to assume a purely Lambertian surface, for example. We do the same in our work. A Lambertian surface is one that reflects light equally into all directions. See Chapter 4 for a closer examination of diffuse surfaces and how the Lambertian BRDF is used in the rendering equation. One result of this constraint is that our input images should not contain any highlights as that would violate the assumptions made about the surface material.

The surfaces in the input images should also have a non-varying albedo. Given an input image, it is difficult to determine how much of the information stored in the intensity values is because of shading effects or lighting, and how much is simply due to color variation on the surface. To intuitively understand this problem, consider a bumpy surface of a rock that has the same color across its surface. Now, if we take a photo of it, all color variation in the image is due to the bumps and cracks of the surface and the way they interact with light. If we then project this photo onto a white plane and take another photo of it, we might be able to reproduce the exact same image. The surface is entirely flat, though, and only the lighting, the projection of the photo, creates the illusion of a bumpy surface. The same can be achieved if we paint the flat surface so that it would reproduce a similar image. Again, the surface would be entirely flat and only the color variation

¹ point, directional and spot lights

on it would create the illusion of a bumpy surface. For our work we are interested in the bumps and cracks and less so in the lighting — the projection in the example — or the varying albedo — the paint in the example. Our constraint of a non-varying albedo excludes the case of a painted surface. We have thus limited the problem domain significantly and can optimize the algorithm accordingly. As said before, our techniques will always solve the given problem as well as possible, but the system is designed to operate on Lambertian surfaces with non-varying albedo under simple lighting conditions. We have also determined how the synthetic images that are compared to the input will be created. Having these images, we need to think about a proper representation through an image descriptor.

7.2 CHOICE OF DESCRIPTOR

We have two main requirements regarding the descriptor. Firstly, it should be able to recognize structures of different sizes and it should also be able to discriminate between rotations and directions in the image, because humans are sensitive to these kinds of features. Secondly, the resulting descriptor vector should be compact enough to be stored efficiently even if millions of descriptors need to be saved to enable interactive performance. Simple statistical moments are not able to fulfill our requirements regarding feature recognition and we will not be able to efficiently store more complicated descriptors like HOGs. Additionally, the HOG descriptor is not suitable for our purpose as it is not able to detect features on different scales, which is crucial in human perception. We have thus chosen the gabor descriptor as briefly described in Section 3.3. We believe to have found a good compromise between the quality of the descriptor and its compactness. The Gabor descriptor has already been successfully employed for similar work by Gieseke et al. [Gie+14].

As for the filter bank to be used, we also follow their suggestion regarding the set of kernels to be included. Regarding scales, six different values for σ are used: $\sigma \in \{0.5, 1, 2, 4, 8, 16\}$. In total the filter bank consists of 374 kernels. We will look at some kernels for which $\sigma = 16$, in the following.

As the aspect ratio will always be 1 for our kernels and θ is set to complete half a turn in 16 steps, the remaining parameters that we can tune are the frequency of the sinusoid λ , as well as the phase offset ψ . For the scale that we look at, three different frequencies are used ($\lambda \in \{32, 64, 128\}$) one of which has an additional phase offset. Fig. 7.1 shows plots of the resulting kernels. The kernels on the other scales are computed in a similar fashion.

Structural information can be well described using the kernels. As can be seen in the plots in Fig. 7.1, the kernels are able to detect rotation and they can be used to detect edges and stripes. In the de-

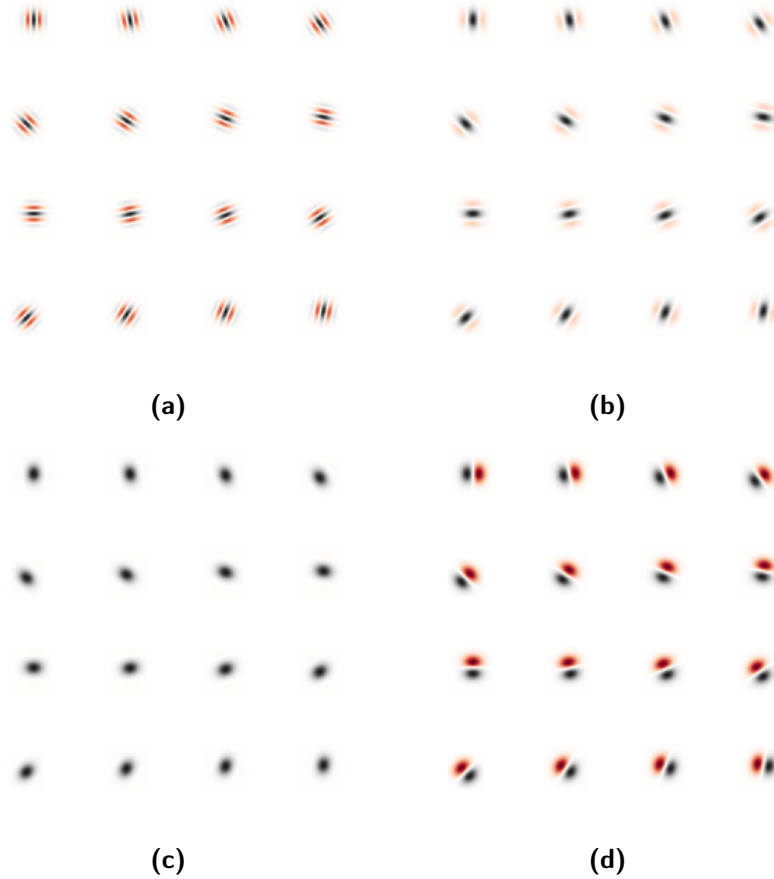


Figure 7.1 – Examples for kernels in the filter bank. Positive values are denoted in black, negative ones in red. (a), (b) and (c) show the kernels with increasing λ . The plot in (d) was done with the same frequency as in (c), but with an additional offset of $\psi = \frac{\pi}{2}$ so that a negative lobe is added.

scriptor vector, we include the mean and standard deviation of the pixel value distribution after convolving the image with the kernel. Note that only pixel locations that do not require crossing the image boundary when convolving with the kernel are included in the distribution. This histogram can be approximated by mean and standard deviation as “convolution implies Gaussianity” [Lag+10a]. The descriptor used for our work is very similar to the one employed by Gieseke et al. with the exception that we can remove the global mean μ_{global} and the global standard deviation σ_{global} . The reason for that is our choice of the space where we compare the images as described in the following.

7.3 SPACE FOR STRUCTURAL COMPARISON

We do not impose particularly tight restrictions regarding the input images. It is only assumed that they are given in sRGB space and

that they follow roughly the assumptions about a dominant lighting direction as outlined before. We do not assume that the image values are scaled to be in a particular range only, though, and we also do not assume anything about the nature of the value distribution from a statistics point of view. The benefit of this approach is that no complex setup for capturing the example images is needed.

Our goal is to compare the input to precomputed renderings in order to find those that represent the closest approximations as well as the according parameters needed. Thus, the input image and the renderings need to be transformed into a common space where comparison is meaningful. Two problems arise here. Firstly, it has to be determined which color space is suited for this comparison and secondly, we need to decide how we transform the image values further to make comparison more meaningful.

7.3.1 *Color Space*

Regarding the color space in which to do the comparison, we could use linear or a space like sRGB or CIELAB. Using linear would put the focus on the actual radiance values captured by the camera and those synthetically created by the renderer. Using sRGB or CIELAB would compare the image values with a focus on how they are perceived in the human visual system.

We have chosen to use linear here, as we are more interested in the actual structure of the surface and their physical representation. Evaluating a color space that is closer to our perception could be an interesting task for the future.

7.3.2 *Remapping of values*

In linear we have a few options regarding the treatment of the image values. We could leave them as they are and do a simple comparison between the descriptors of input image and synthetic image. This is not ideal, because we have some assumptions as outlined before that heavily influence the shape of our renderings. To start with, we have assumed the intensity of the directional light source to be 1. We can not assume that this is really the case for the input images, though, and if our assumption is not met here, there will be an offset in all the image values of the input. Simply comparing input and rendering will lead to incorrect results for that reason, as a structure might accidentally produce similar radiance values as the input, without being a good approximation. Secondly, we assume the synthetic surface to have an albedo of 1 — or plain white — at the moment and ignore ambient light entirely. Again, the surface in the input image will not adhere to these constraints. For that reason the mean of the input might be entirely different than that of the rendering, even though the

approximated surface structure is a good match. Bad matches will be the result.

One option would be to scale both input and rendering to contain image values between 0 and 1. Eq. (7.1) shows how this transformation can be done for all image values i to get new values i^* .

$$i^* = (i - a) \cdot \frac{1}{b - a} \quad (7.1)$$

The values a and b are assumed to be the smallest and the biggest occurring values in the image. While this improves the similarity of the images, it will still not adjust the mean correctly, for example. Also, outliers in the input (spot noise, for instance) can dramatically affect the way the image is transformed. Gieseke et al. implicitly use a space for their work where $I(x, y) \in [0, 1], \forall (x, y) \in \Omega$, because their procedural textures are implemented to output values that are scaled to fit into that range. As a consequence their input images need to be transformed into this space when loaded so that they are comparable to the procedurals. We believe that this approach is not ideal if shading is involved in the creation of the synthetic images. The textures that Gieseke et al. generate are very similar in their intensity value distribution for varying parameters. A relatively smooth distance function is the result. The renderings that we compare to the input image differ hugely due to different lighting directions and do not change smoothly with changing parameters. It is for that reason that we need to enforce a robust space for the comparisons. The wide range of results that we obtain when rendering structures under varying lighting conditions increases the probability of finding a bad surface match with a small distance to the input only because of random offsets based on the image capturing conditions.

At the moment, we are only interested in the structure of the surface so that we should remove all other differences, ideally. Following this line of thinking, the images should be the same if we remove all structure. We could do that through linear diffusion, for example, which is very closely related to convolution with a Gaussian kernel. By applying linear diffusion to the image, we are progressively redistributing the mass (the image values in this case) to be equal everywhere. For images this is equivalent to setting all pixel values to the mean of the image value distribution. In order to be able to isolate the structure well, the mean of the images should be equal for this reason. The standard deviation of the value distribution should be identical as well which can be understood intuitively by regarding it as a measure of contrast in the image.

We have thus decided to transform the images in a way that adjusts their mean and standard deviation. The mean is set to 0 and the standard deviation to 1. Note that we can only do that since we have chosen not to use simple first order statistics as the descriptor. If we had, two of the entries there would be equal across all images and

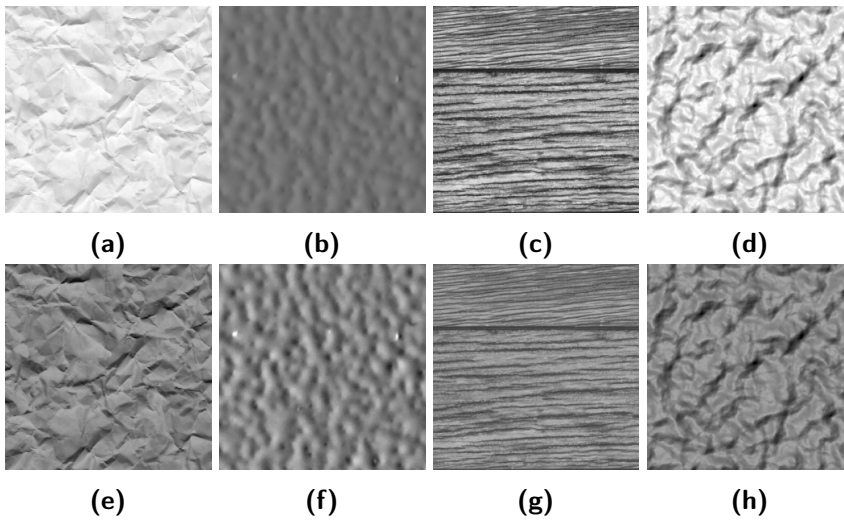


Figure 7.2 – Images and a rendering in sRGB space (a-d) and in our comparison space (e-h). Note how the remaining differences in the images are mostly due to structural differences of the surface.

renderings since we explicitly manipulate the images so that these values are the same. This is another reason for using a more complicated descriptor than first order statistics as it allows a little more flexibility when it comes to transforming the image values. The process of explicitly setting mean and variance is used heavily in color transfer [Rei+01] and is described in Section 9.2 more closely. As mentioned before, this process also allows us to simplify the descriptor a little, compared to the work by Gieseke et al. [Gie+14], as we do not need to store the global mean μ_{global} and the global standard deviation σ_{global} anymore as they are set to 0 and 1 respectively anyway.

We will refer to this space as *comparison space* and show examples after remapping them to be useful for viewing purposes. Fig. 7.2 shows examples of this space for possible input images and a rendering.

7.4 TEXTURE MODELS

In terms of models for the height fields, we support a few different procedural textures. We then add scene related parameters to the parameter space of the texture itself. These scene parameters are the main difference to the texture models used by Gieseke et al. [Gie+14]. Fig. 7.3 shows available textures that were generated using their default parameters.

7.4.1 Light Parameter

One part of the scene is the lighting. As we have said before, we use a directional light source with an intensity of 1 for lighting the surface. The position of the light is of importance for the resulting shading.

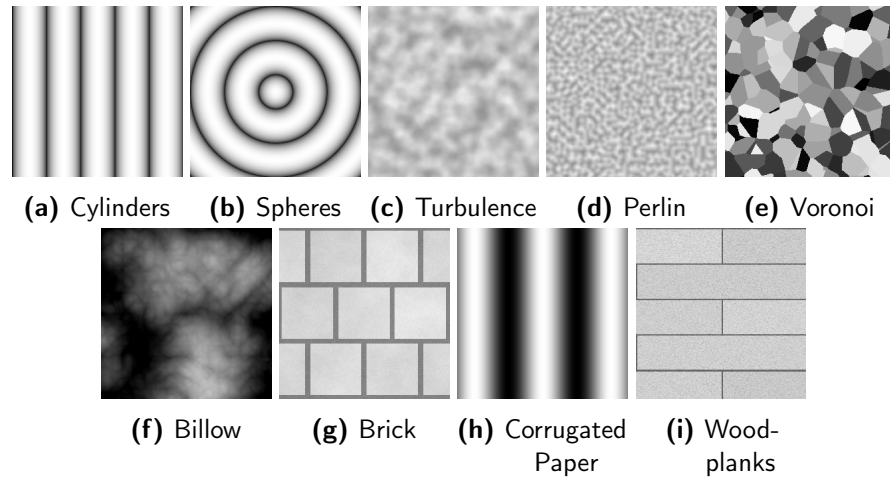


Figure 7.3 – Set of procedural textures that can be used as height maps. They are shown according to their default parameters here.

The light is assumed to be located on the unit hemisphere above the surface which lets us use the position as the direction in the case of a directional light. Fig. 7.4 shows all available positions that are allowed in our system. The positions were created using an iterative method based on electrostatic charges to achieve uniform spacing [Fuc+07]. The position of the lights are determined via an id which allows us to model this particular dimension of the parameter space as a natural number $id \in [0, 63]$ for the 64 different light locations shown in Fig. 7.4.

7.4.2 Amplitude Parameter

Additionally, the amplitude of the relief needs to be parametrized. Different amplitudes will result in different renderings and are directly related to the contrast of the image and the standard deviation of the resulting image value distributions. Low amplitudes results in little contrast and vice versa. It has a strong influence on the resulting image descriptor for this reason. Modeling the amplitude via a special parameter also allows a wider range of reliefs that are able to approximate real surfaces better. Examples can be seen in Figure 7.5.

7.5 TARGET IMAGES

The images that are used as input should roughly follow the assumptions as described before. Uniform albedo and a dominant lighting direction are the main requirements. Fig. 7.2 shows three possible examples (a–c) that will be used later on. As mentioned before, images that violate these assumptions can be used as well, but they might lead to results that are not ideal.

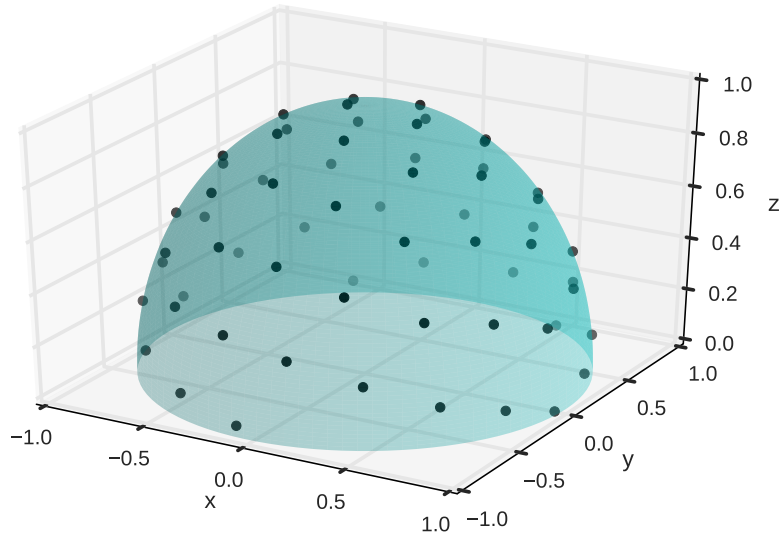


Figure 7.4 – The locations of the directional light sources used for lighting the surface. Positioned on a unit hemisphere, the location corresponds directly to the direction for a directional light source.

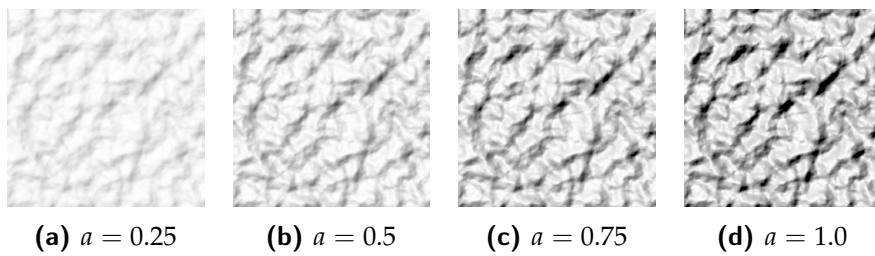


Figure 7.5 – Renderings generated with the same procedural height map but different values for the amplitude a .

FINDING STRUCTURAL MATCHES

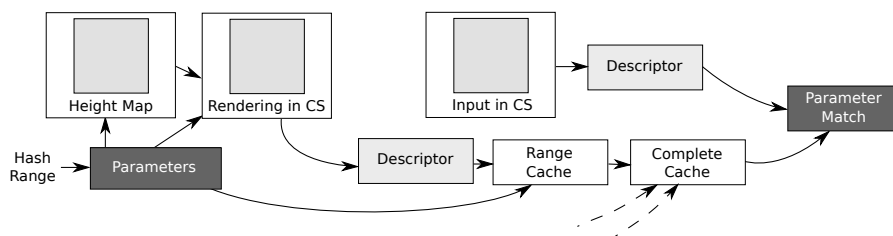


Figure 8.1 – Pipeline Overview. Input images and renderings created from a procedural height map are compared in comparison space (CS) to find good matches.

In this chapter we will look at how to find the parameters for a procedural displacement map in order to approximate the surface structure shown in the input image as closely as possible. To this end, we need to discuss how the reliefs that we create procedurally are rendered and how the resulting image descriptors are stored and used later on. See Fig. 8.1 for an overview of the pipeline.

8.1 RENDERINGS

The following steps are necessary to create renderings. First, the set of parameters to be used for the rendering needs to be created. Each texture model provides a hashing mechanism that allows mapping from a simple integer value to a parameter list. That way it is easy to run the renderings sequentially and thereby sample the entire parameter space. Note that the model itself defines how sparsely each dimension is sampled.

Once the set of parameters is generated, the height map can be created. Then the light *id* in the parameter list is converted to a light direction. A normal map is derived from the height map and is used by a renderer in conjunction with the lighting direction to evaluate the Lambertian shading. At this point a full rendering is generated and the next step in the pipeline can be started. See Fig. 8.2 for some examples of renderings created by the system.

8.2 DESCRIPTORS AND CACHES

To generate the descriptor vector, the rendering is transformed into the comparison space defined in 7.3. As described in 7.2, a descriptor is computed through convolving the rendering with the kernels of a

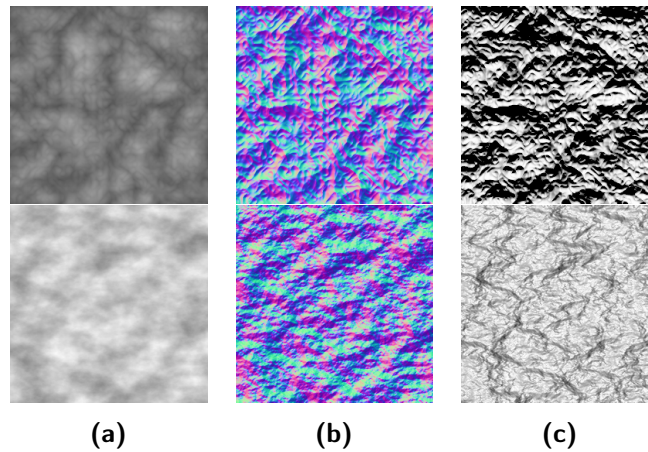


Figure 8.2 – Example renderings. Shown are the relief (a), the respective normal (b) and the final rendering (c).

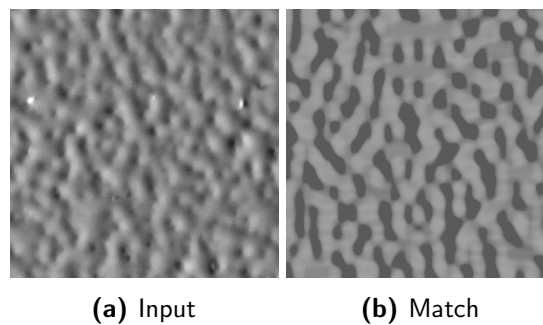


Figure 8.3 – A match found for the input. Both are shown in comparison space. The *Perlin* model is used for the relief.

predefined filter bank. To make this computation more efficient, the convolutions are performed in frequency domain using Fourier transformation [Smi+97]. The process is actually very similar to the work done by Gieseke et al. [Gie+14], with the exception that we transform the image into a different space before computing the descriptor.

After the descriptor vector is computed, it is stored into a cache file in which we also store the set of parameters needed to create the rendering. To speed up creating the caches, the descriptor computation can be distributed onto multiple systems each handling a range of hashes. Results are recombined to a complete cache file later on.

With the cache in place, we can start looking for matches once we get an input image that we want to approximate procedurally. The descriptor for the input needs to be created in the same way as for the renderings. It is then compared to every entry in the cache in order to find the most suitable set of parameters. The distance metric described in 3.3.3 is used to determine the closeness of a match. At this point the problem of finding a structural match is solved and we can look into improving the overall match regarding other surface qualities. Fig. 9.5 shows an example of a match in comparison space.

FINDING SHADING PARAMETERS

In the preceding chapter we have described how to find structural matches that satisfy our requirements in terms of structural similarity to the input image. We will now have a look at finding according shading parameters that allow approximating the material depicted in the input even further. What constitutes a close match will be derived in the following and an intuition how shading comes into play will be built.

Figure 9.1 shows the structural match that we looked at in Chapter 8 already. It also includes the actual input image and the height map rendered using default shading. Comparing these two we can see that some shading will be needed to match the material more closely if the light direction is assumed to be a good fit.

So far the quality of a match was determined by looking at the distance between its descriptor and the descriptor for the input image. Small distances denote similarity which is what we are aiming for. Let us continue this train of thought for a minute, although it will not lead to a method for deriving the shading parameters. It will enable us to gain some insight of how shading influences the statistical properties of an intensity value distribution, which will be helpful for the discussion that follows.

9.1 USING THE DESCRIPTOR

We could try to minimize the distances between the descriptors even further and do that by using the coefficients k_a and k_d as used in the shading model from Chapter 4, reiterated in Eq. (9.1).

$$I(x, y) = k_a + k_d \cdot \cos \theta_{x,y} \quad (9.1)$$

The model contains the parameter k_a which is used to avoid black shadows. These do not really occur in natural images due to light

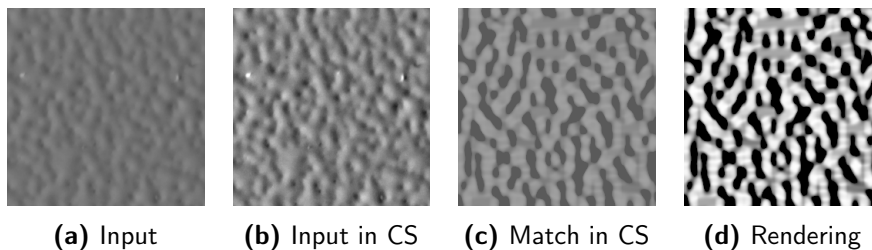


Figure 9.1 – Example of a structural match for a given input including the resulting rendering. Both are also shown in comparison space (CS).

bouncing around in the scene and illuminating shadowed areas. Before it was possible to simulate all these lightpaths, using an ambient term k_a was a simple ad-hoc approach to improve the look of an image. We will keep using this simple model in the following and fit k_a and k_d . We propose its use as it is compact and easy to understand. The ambient lighting that k_a describes encapsulates all the complex lighting effects that are necessary to generate a photorealistic image, but can be combined here as additional light that is not coming from our main directional light. It thus represents interreflections on the surface as well as light coming from other light sources, e. g. the sky or multiple soft light sources without strong directionality. Since we are primarily interested in the material depicted in the input image and not in the light setup used to capture it, we simply combine all these effects of minor importance into a simple additive constant k_a which of course is merely an approximation. Our shaded image is modeled as shown in Eq. (9.1) for that reason. Note that k_a and k_d are simplified to be non-varying in regard to different locations on the surface.

For the optimization we will have to find those coefficients k_a and k_d that reduce the differences between the individual entries in the descriptors. The descriptors consist of means and standard deviations computed after convolving an image with a filter bank consisting of M kernels k . We should thus look at the effects that the shading coefficients have on these values. The mean $(I * k)_\mu$ for an image with N pixels that contain the results of a convolution can be computed as shown in Eq. (9.2).

$$(I * k)_\mu = \frac{1}{N} \sum_{i=1}^N (I * k)_i \quad (9.2)$$

$$= \frac{1}{N} \sum_{i=1}^N \left(\sum_{l=1}^K (k_a + k_d \cdot \cos \theta_{i,l}) \cdot k_l \right) \quad (9.3)$$

Eq. (9.3) shows the convolution with the simple shading model plugged in. The discrete kernel that is used here has K entries. For the default case with $k_a = 0$ and $k_d = 1$, the computation of the average simplifies to

$$(I * k)_{\bar{\mu}} = \frac{1}{N} \sum_{i=1}^N \left(\sum_{l=1}^K \cos \theta_{i,l} \cdot k_l \right), \quad (9.4)$$

which is what we have stored in the precomputed descriptors. Using some simple transformations we can show that (9.3) is equal to

$$(I * k)_\mu = k_a \sum_{l=1}^K k_l + k_d \cdot (I * k)_{\bar{\mu}} \quad (9.5)$$

so that we can avoid convolutions and directly manipulate the entries denoting averages in the descriptor to find the values for a descriptor

that describes a shaded image. Note that we still need the sums of the kernel entries used for the convolution.

To show that the entire descriptor can be transformed without computing convolutions, we also need to show that standard deviations can be transformed in a similar manner. Following the same argumentation, it can be shown that the standard deviation stored in the descriptor for images with default shading coefficients $k_a = 0$ and $k_d = 1$ can be computed like shown in Eq. (9.6).

$$(I * k)_{\bar{\sigma}} = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(\sum_{l=1}^K \cos \theta_{i,l} \cdot k_l - \frac{1}{N} \sum_{j=1}^N \left(\sum_{l=1}^K \cos \theta_{j,l} \cdot k_l \right) \right)^2} \quad (9.6)$$

It can then be shown that the entries denoting standard deviations in a descriptor for an image with non-default shading coefficients can be computed as

$$(I * k)_{\sigma} = k_d \cdot (I * k)_{\bar{\sigma}}, \quad (9.7)$$

which proves that the entire descriptor can be easily transformed to account for changes in the shading if this particular shading model is used.

Finding the least-squares solution to the linear equation in Eq. (9.8) thus gives us the coefficients k_a and k_d that minimize the distance between the descriptors.

$$\begin{bmatrix} \Sigma_1 & (I * k_1)_{\bar{\mu}} \\ 0 & (I * k_1)_{\bar{\sigma}} \\ \vdots & \vdots \\ \Sigma_M & (I * k_M)_{\bar{\mu}} \\ 0 & (I * k_M)_{\bar{\sigma}} \end{bmatrix} \times \begin{bmatrix} k_a \\ k_d \end{bmatrix} = \begin{bmatrix} (I * k_1)_{\mu} \\ (I * k_1)_{\sigma} \\ \vdots \\ (I * k_M)_{\mu} \\ (I * k_M)_{\sigma} \end{bmatrix} \quad (9.8)$$

The sums of the entries in the M kernels are denoted as Σ_1 to Σ_M here.

We have thus seen that the descriptors can be used directly for finding optimal shading coefficients and we have also seen that these coefficients influence the mean and the standard deviation of an image significantly. They can be used to create closer matches between the stored descriptor of a rendering and that of a given input image. The problem with this approach for our application is the space that we have chosen for structural matches. Matches are computed in a space where differences in the mean and the standard deviation are explicitly removed to be able to compare structures in the image. If our descriptors would be computed in a different space, the one used by Gieseke et al. [Gie+14] for example, the outlined approach would be very useful. In our case, remaining differences in the descriptors are not because of global offsets or scaling factors, but purely because of structural differences which compute different results when being

convolved with a kernel. We can rule out using the descriptors for finding the shading parameters entirely for this matter and need to switch to a two-step process, because the computations to find structural matches proved to be useless for the upcoming task. If comparing descriptors is not a suitable measure for the quality of matches anymore and the color space used so far is not helpful, we will have to solve these problems in the following and introduce new methods. We will start by putting forth an argument regarding color spaces.

9.2 COLOR SPACES AND SIMILARITY

As described in Chapter 2 when we talked about color spaces, all shading needs to take place in linear color space. The reason is the linear nature of light transport [PH04]. Color spaces like sRGB apply a transformation that makes images look correct on screens which themselves apply a display gamma. This additional transformation in spaces like sRGB makes it impossible to compute proper physically correct results in a space other than linear. It is for that reason that our computations that retrieve the shading coefficients to match our rendering to the input image need to take place in linear color space, if we want the retrieved k_a and k_d to have a physical meaning. The retrieved shading parameters would not work in a renderer, if they were not constraint to work correctly in linear. Having found a color space that should be used for determining the distance between images, we still need to define what actually constitutes this distance.

If we interpret the general difference between histograms as mostly differences in their mean μ and their standard deviation σ , we can see that a multiplicative change and an additive offset can be used to adjust the histograms to each other. Doing these adjustments is common when transferring color between images with the goal of matching the look and feel of an image to a target [Rei+01]. Since we are ultimately interested in finding a procedural material that looks closest to the input after being rendered, it seems legitimate to apply this measure of “closeness” based on μ and σ . In the previous section we have seen that we can interpret the shading as changing the mean and standard deviation of an image, k_a thus being the additive offset needed and k_d the multiplicative factor.

9.2.1 *Fitting in Linear*

As shown in Figure 9.2, we will have to convert the input to linear to have a linear version that we can compare to our rendering. Once that is done we can start adjusting the histograms according to our requirements. We have previously defined our distance according to the differences between the mean μ_i of the input image and μ_o of the output image, and the difference between the respective standard

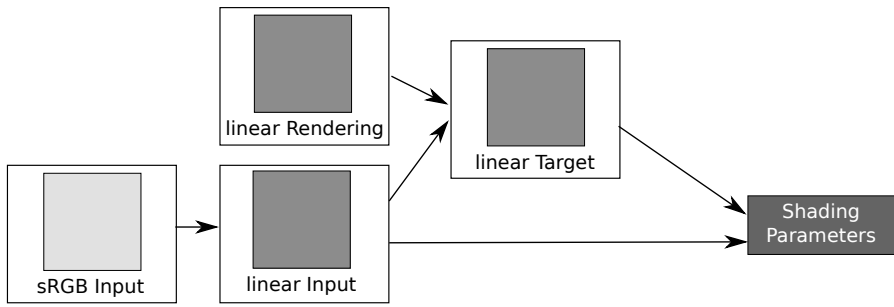


Figure 9.2 – Process for matching the shading parameters. The linear target is created by adjusting the mean and standard deviation of the linear rendering based on the histogram of the input image. This target is used to optimize the shading of the linear rendering.

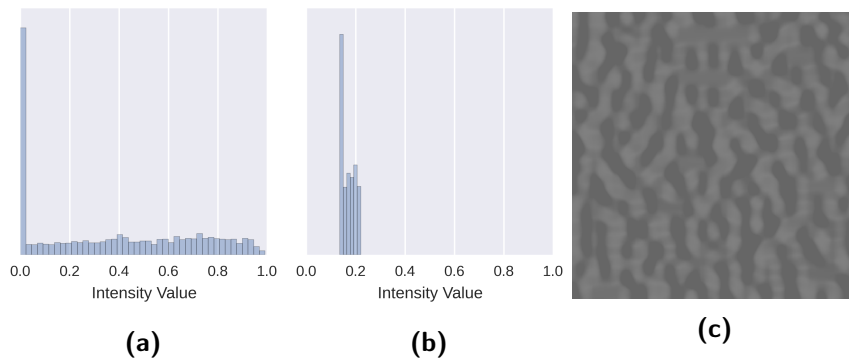


Figure 9.3 – Adjusting the linear rendering. (a) shows the rendering’s histogram before and (b) after adjusting its mean and standard deviation according to the input image. Resulting image (c) is shown in sRGB here.

deviations σ_i and σ_o . Ideally this distance is 0 and we can enforce this property as described by Reinhard et al. in [Rei+01]. A similar process was applied for the transformation into comparison space. The following formula shows how to scale and shift each individual linear data point d_i in our rendering in order to have data points d_i^* whose distribution matches that of the input image regarding its first order statistics.

$$d_i^* = (d_i - \mu_o) \cdot \frac{\sigma_i}{\sigma_o} + \mu_i \tag{9.9}$$

Figure 9.3 shows the histogram of the rendering from Figure 9.1 before and after adjusting according to the formula in Eq. (9.9). The transformed rendering can be seen in Figure 9.3c. We will call this image *linear target*.

At this point we have an image that was adjusted to have the same mean and standard deviation as the linear version of the input. The next task will be to find the shading parameters that transform the unshaded linear rendering as closely as possible to this linear target.

We will still use the model shown in (9.1), but we deviate from this classical model slightly and interpret k_a as the intensity of an additional ambient light source instead. The only difference in this interpretation is that k_d is factored into the ambient response which is important for color images, as unlit areas of the image would appear gray otherwise.

We can now define a cost function that allows us to penalize deviations of the rendering $R(k_a, k_d)$ from the ideal target T :

$$c(T, R(k_d, k_d)) = \sum_{i=0}^N (T_i - R(k_a, k_d)_i)^2 \quad (9.10)$$

Images are assumed to be made up of $N = w \times h$ pixels. Optimization methods can then yield the shading parameters needed to approximate the target as closely as possible.

9.2.2 Fitting in sRGB

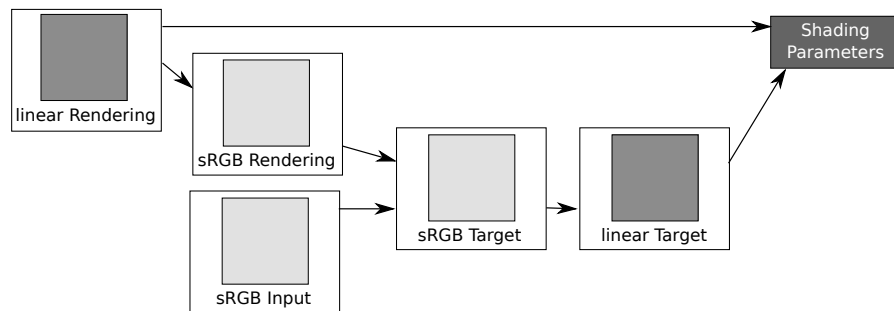


Figure 9.4 – Process for matching the shading parameters via an sRGB match. The linear target is created by adjusting the mean and standard deviation of the rendering in sRGB space based on the given input image. Optimization is then done as before.

As an alternative to adjusting the mean and standard deviation of the rendering in linear space, a linear target image can be generated by matching in sRGB space first. When matching in linear space as done in the last section, the image value distribution is usually changed drastically to match the target. It is not guaranteed that the same image value distribution transformed to sRGB will ideally match the input image when that is transformed to sRGB as well.

An example for which the method described in the last section gives results that are not ideal can be seen in Fig. 9.5. Here the same input image as before is color corrected to give results that show the problem more clearly. The image is darkened a little and the contrast is enhanced. It can be observed that the resulting mean is a better match when the adjustments are made in sRGB. In general, renderings created with parameters found through matching in linear space tend to have a bigger standard deviation and therefore display

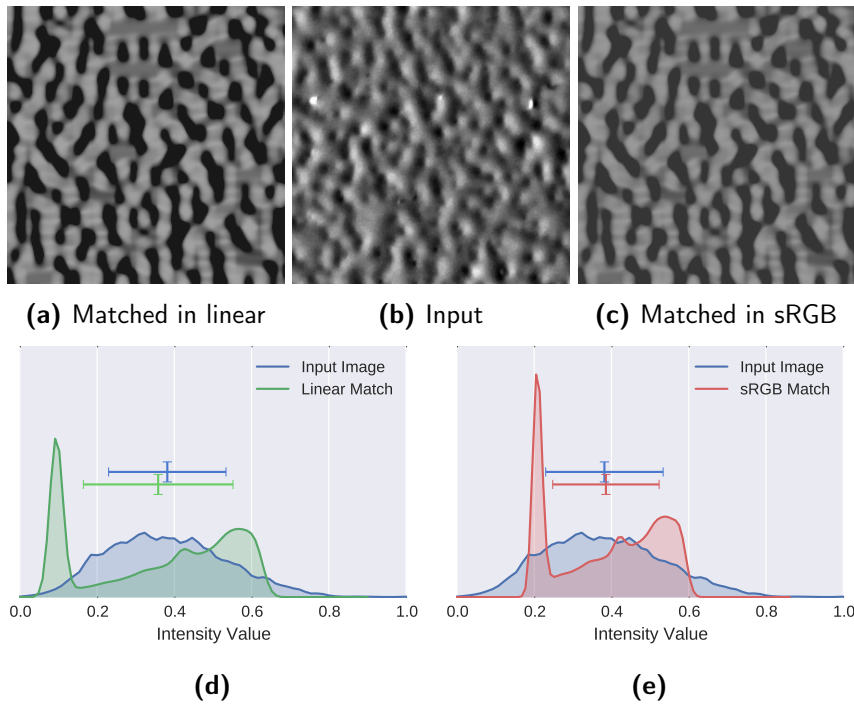


Figure 9.5 – Example of shading parameters found through a match in linear space that are inferior to parameters found in sRGB space when displayed in sRGB. The distribution plots in (d) and (e) include μ and σ . They show greater correspondence for the match found using sRGB regarding first order statistics.

a higher contrast. This can be seen in Fig. 9.5, where 9.5a shows a much wider range of image values than the match found in sRGB space shown in 9.5c. Looking at the respective histograms in Fig. 9.5d and Fig. 9.5e and the values for μ and σ , this impression is confirmed. In cases where it is important that the materials match as well as possible in a particular viewing color space, that space should be used for matching first order statistics.

9.3 COLOR

So far we have only looked at grayscale images with shading parameters k_a and k_d that are simple scalars. In this section we extend what we have done so far to include color.

Here, our interpretation of k_a becomes important as it should be kept a simple scalar. The reasoning is the difficulty of determining the color of a surface, if the spectrum of the incoming light is unknown. By assuming perfectly white light with an intensity modeled by a scalar, we can infer the surface color. Following this assumption, we can simply repeat the process outlined for grayscale images for the three channels of a color image. A result can be seen in Fig. 9.6.

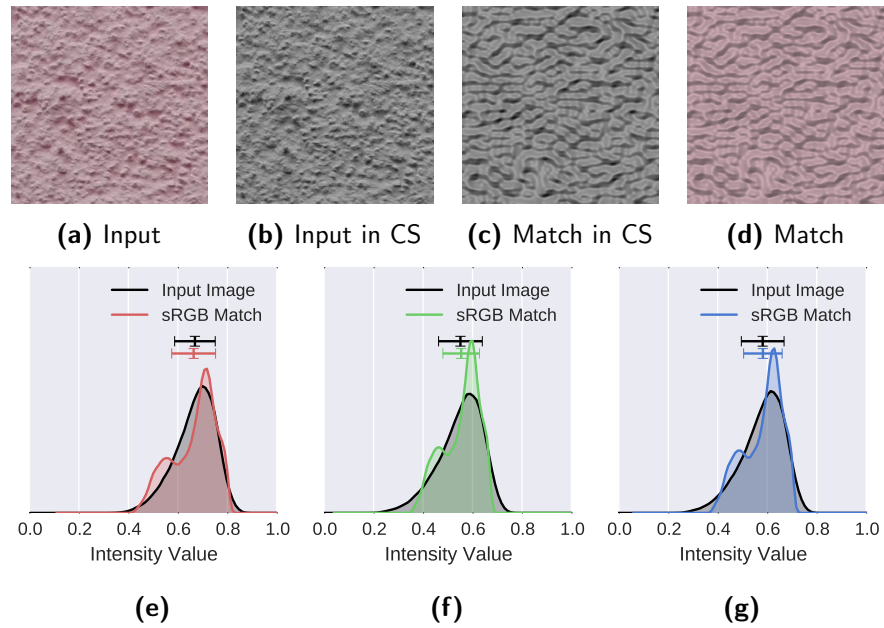


Figure 9.6 – Example of a match that considers surface color.

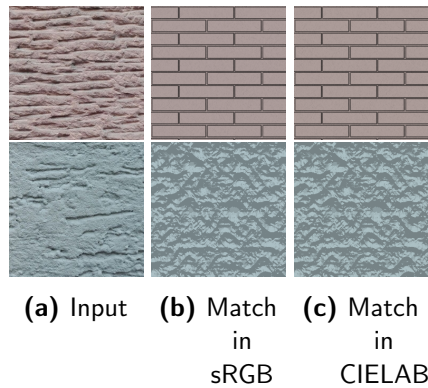


Figure 9.7 – Examples for matches created using sRGB and CIELAB. The examples were created using the *Brick* and *Billow* procedural.

In Section 9.2 we have already examined the effect of changing the color space in which μ and σ are matched. Following this idea, we consider now whether better results can be achieved when using CIELAB instead of sRGB for this matching. Examples can be seen in Fig. 9.7. To properly evaluate the actual differences regarding which image is considered a better match by a human observer, a study has to be conducted. Since the differences are small enough to not be immediately noticeably in most cases, we will at this point conclude that sRGB is suitable for our purposes and ignore CIELAB as the transformations include an additional performance hit. In our tests we treated the color channels independently as we did when using sRGB.

IMPLEMENTATION DETAILS

We have developed techniques to retrieve the parameters for a procedural material. In this chapter we will look at the implementation specific details of our work.

Most of the code is written in Python and makes extensive use of libraries like NumPy¹ and SciPy² to achieve interactive performance. Some of the implications regarding cache sizes and retrieval performances will be discussed here. All results are reported for an Intel® Core™ i7-3630QM CPU @ 4 × 2.40GHz.

All our image types are based on two-dimensional NumPy arrays for grayscale images or three-dimensional arrays for color images. The actual creation of data for the procedural textures is done in C++. The respective code is wrapped using Cython³. A part of the library for creating procedural height fields is taken from Gieseke et al. [Gie+14] and the rest is a wrapper for textures provided by “lib-noise”⁴, a C++ library concerned with creating procedural textures.

10.1 CACHES AND DISK SPACE

Since we are creating a descriptor for every sample in the respective parameter spaces and store all of them for later use, it is important to look at the resulting disk space requirements. In order to retrieve results quickly, caching is needed as computing descriptors on-the-fly for all samples would not allow interactivity. Table 10.1 shows the resulting sizes for all classes of procedural textures used as well as the number of entries stored. Please note that the parameters used to create the respective height maps are stored alongside the descriptors and contribute to the given sizes. Gieseke et al. [Gie+14] report slightly lower cache sizes. They use a custom cache format that is implemented in C++, while we rely on NumPy’s file storage capabilities. All descriptors and respective parameters are stored as two large two-dimensional arrays into an npz file. These files are zipped archives and contain npy files that store the arrays. One of the goals of the format is system independence, which requires storing the shape of the arrays and data type information into the file [Ker07].

¹ Information can be found in [VDWCV11] or [Oli06], for example

² See [JOP+01] for further information.

³ Behnel et al. [Beh+11] provide an introduction

⁴ Available from [Bevo3]

TEXTURE CLASS	ENTRIES	COMPRESSED	UNCOMPRESSED
Billow	1151975	5.1 GB	6.5 GB
Brick	1115125	4.9 GB	6.3 GB
CorrugatedPaper	77550	249 MB	447 MB
Cylinders	9975	32 MB	58 MB
Perlin	169325	749 MB	975 MB
Spheres	38775	173 MB	223 MB
Turbulence	532200	2.2 GB	3.0 GB
Voronoi	511100	1.9 GB	2.9 GB
Woodplanks	139375	627 MB	803 MB

Table 1 – Cache sizes for the procedural textures that our system implements.

10.2 RETRIEVAL AND FITTING PERFORMANCE

To find matches, the descriptor of the input image has to be computed and compared to all entries in the caches using the developed distance metric. In this section we analyze the performance of these processes. We will use the *Turbulence* procedural in the following for our measurements. The size of its caches is average compared to those of the other models. Fig. 10.1 shows the input image and the matches used for the performance tests.

10.2.1 Computing the Descriptor

In order to compute the descriptor, convolutions with the filter bank need to be performed. Our implementation uses the SciPy module for this task. Like Gieseke et al. [Gie+14], we compute the convolutions using frequency space to improve performance. The function `scipy.signal.fftconvolve` is used for this purpose and allows computing the descriptor for 374 kernels in a little over two seconds. Note that the mean and standard deviation for the result after convolution are computed on the valid image region only. Pixels for which the kernel crosses the image boundaries are omitted in these calculations.

10.2.2 Parameter Retrieval

Regarding the retrieval of parameters for the structural match, the system used by Gieseke et al. [Gie+14] performs slightly better than ours when using a single core. It is still possible to get a result within about 11 seconds for a cache as big as the one for the *Turbulence* model. Multiprocessing can be used to process parts of the cache in parallel. Comparing our result of multiprocessed retrieval to the timings re-

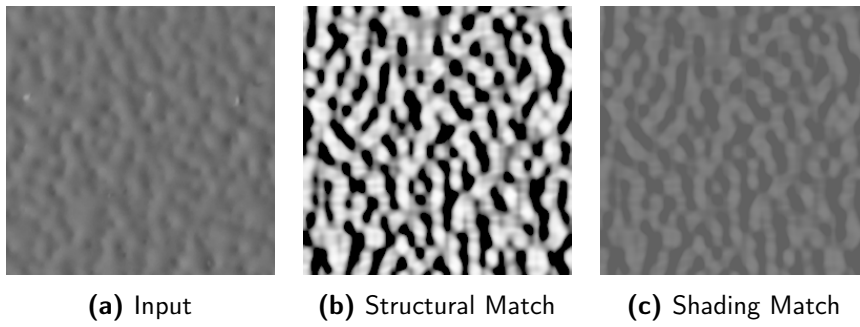


Figure 10.1 – Input and matches for performance measurements.

PROCESS	RUNTIME IN SECONDS
Computing descriptor	2.237
Retrieval from Compressed Cache	17.844
Retrieval from Uncompressed Cache	11.247
Retrieval from Uncompressed Cache (multiprocessing)	3.202
Optimization of Shading Parameters	0.340

Table 2 – Performance measurements for computing the descriptor and for parameter retrieval. The average of three runs is given in seconds. The model *Turbulence* is used for the matches. Results can be seen in Fig. 10.1.

ported by Gieseke et al. [Gie+14], our system performs slightly better. Our system additionally allows using compressed caches in order to save disk space. The retrieval performance is not sufficient for interactive performance, though, as the caches need to be uncompressed before computations can start.

Retrieving optimal shading parameters can be done interactively. Our retrieval mechanism is based on functionality from the SciPy module and uses the function `scipy.optimize.minimize` for minimization. Internally, SciPy uses the L-BFGS-B algorithm, which allows simple bounds for the results. The parameter k_a and all components of k_d are constraint to be in $[0, 1]$. These bounds ensure physical plausibility.

DISCUSSION OF RESULTS

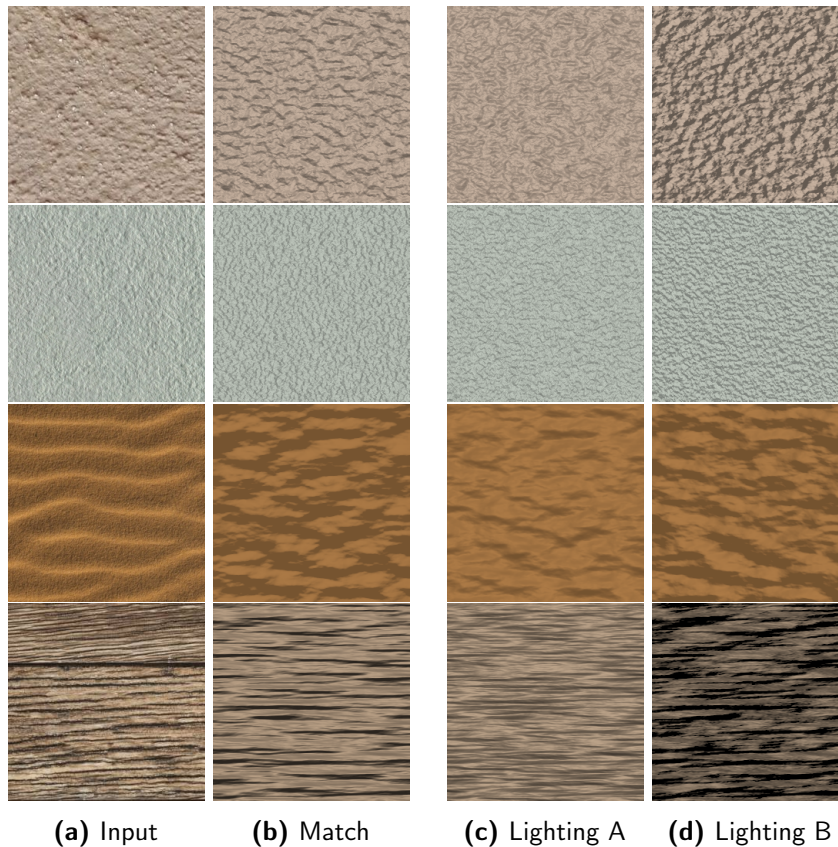


Figure 11.1 – Results in (b) are obtained using the developed method. Included are renderings of the surface under different lighting conditions in (c) and (d). Shown are matches for the model *Turbulence*.

The methods outlined in the preceding chapters are developed to automatically create a procedural material from an example photograph. The parameters of a suitable procedural displacement map are retrieved and the shading parameters for a simple Lambertian shading model estimated. In this chapter we will discuss results obtained by the system.

As can be seen in Fig. 11.1, good matches are retrieved using the developed techniques. Using different lighting directions shows that the surface structure is approximated well. The albedo that was found leads to reflection properties that match the input. Fig. 11.2 shows plots of the resulting geometric structures.

We can thus conclude that the work by Gieseke et al. [Gie+14] can be extended to also include shading information in order to not only match diffuse textures but also surface structure. The choice of the de-

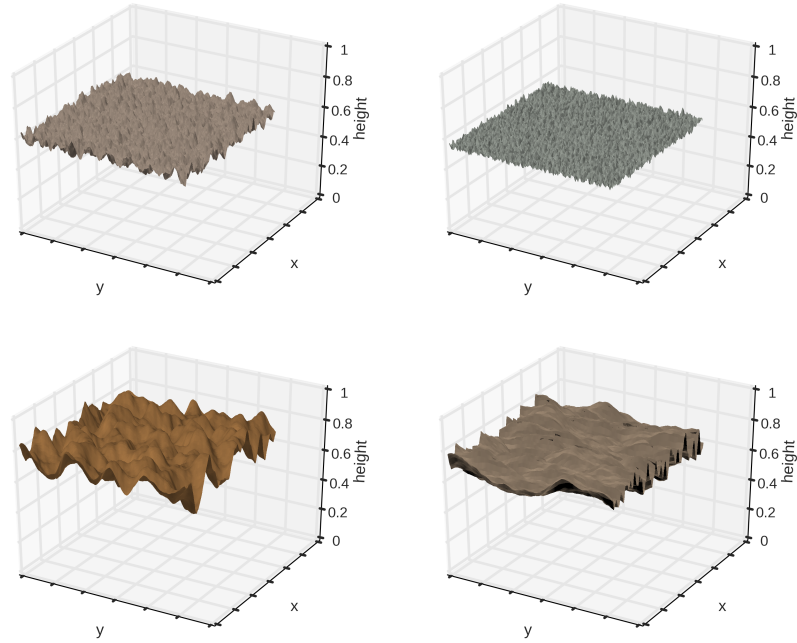


Figure 11.2 – Surface plots for the examples in Fig. 11.1

scriptor and the filter bank is suitable for this task and the resulting descriptors are compact enough to retrieve surface structure interactively. A complete overview of the retrieved matches for all implemented procedural models is shown in Table 3 for grayscale input images and in Table 4 for color images.

The distances shown in the tables allow to evaluate whether a match that is designated good by the system is also a good match according to our own perception. We discuss the results in row (d) of Table 3 as an example. Two matches stand out based on the distances. Both *Corrugated Paper* and *Cylinders* are able to approximate the input better than the other models. Their distances to the input are very similar, which is reasonable as it would be difficult for a human observer to determine the better match as well. The matches that follow in this ranking are able to approximate the scale of the features well, but are not able to create the stretch in y-direction. Following matches have too many horizontal features, too much small scale noise or no structure at all. The other rows in Table 3 exhibit similar behavior, but favor stochastic patterns instead. Our method is not limited to particular procedurals that fulfill certain criteria, regarding their power spectrum, for example. Because of that it is more general than related methods like the work by Gilet and Dischler [GD10]. Regular textures can be used as well as stochastic ones. The results in Table 4 confirm this impression and show that the retrieved albedo is suitable to replicate a surface’s reflectance properties.

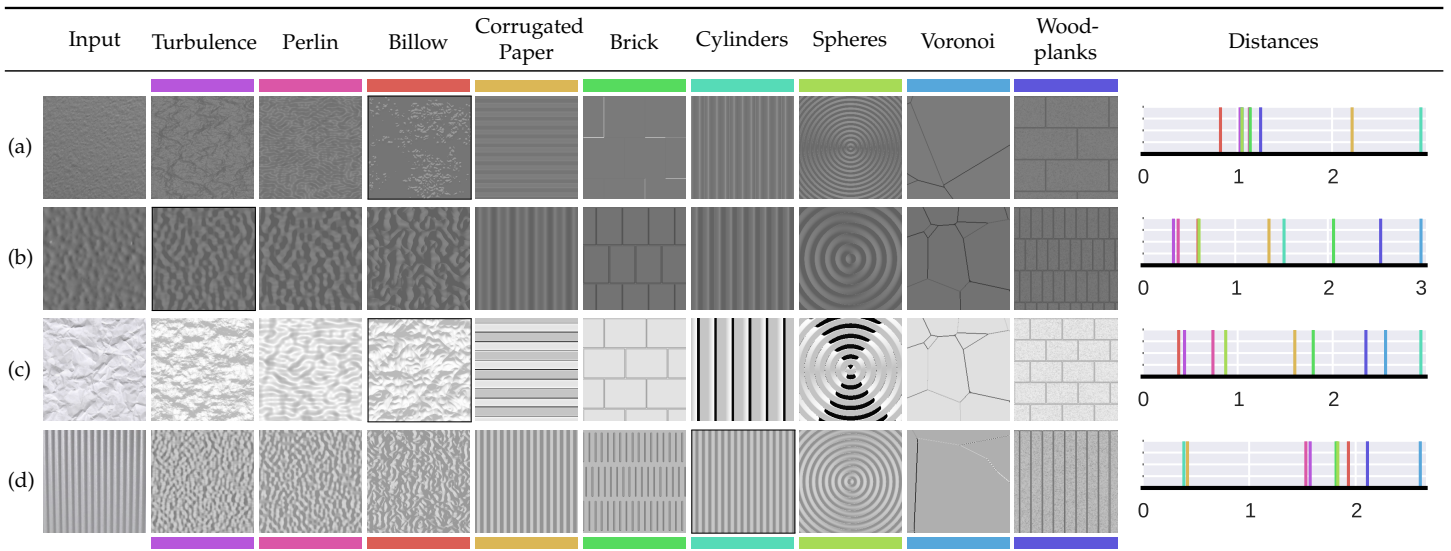


Table 3 – Complete list of matches for all implemented procedural models with grayscale input images. The distance plots show the distance between the descriptors of input image and matches. It is a measure of how well the model performed for the given input. The best match according to the distance metric is framed.

Input images are approximated to the extend possible, depending on the model that is used. This can be seen in Fig. 11.3, for example. A simple change in the input image can lead to much better results as it will allow the system to tune a particular parameter that can help to achieve the desired results. In the example in Fig. 11.3, a simple rotation allows employing a parameter that is used for stretching the procedural in x-direction to achieve elongated structures. A similar parameter for vertical features does not exist in the model.

Noticeable color artifacts are visible in some results in Table 4. They occur because the renderings that show the procedural materials are clamped to only allow image values $I(x, y, c) \in [0, 1]$. If one or more of the channels are clamped, the hue of the color is shifted. Using high dynamic range images with tone mapping would solve the problem by rescaling the intensity values before clamping. We are interested in the geometric structures themselves and decided against generating computationally more costly previews.

For a few inputs, the results do not match our own perception. We created a few examples deliberately by turning some input images by 90° . Our models do not have parameter settings that can generate vertical structures, which makes it impossible for the system to obtain very good matches. The rows (j) and (m) in Table 4 show mismatches, for example. This seems to be a problem mostly with how the distance function is computed as better perceptual matches exist but are estimated to have a larger distance to the input image.

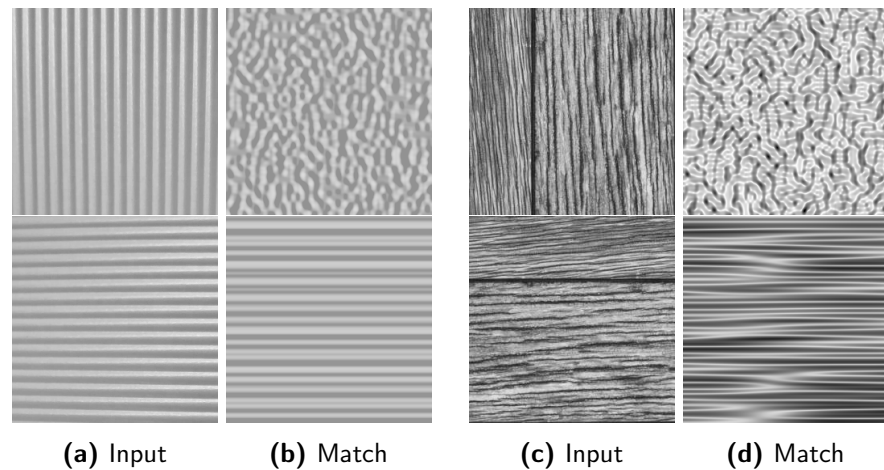


Figure 11.3 – Effect of rotation. The *Perlin* model has a stretch parameter in our implementation that is able to create horizontal streaks, but can not match structures that are more vertical in nature.

11.1 LIGHTING

The results also show that the lighting direction is usually matched well. Although there are cases in which different lighting directions can lead to similar output images, the dominant lighting direction can be retrieved using our method in the majority of situations. Fig. 11.4 shows an example of a structural match that is not particularly good because of limitations in the model. The parameters are set to make the match as close as possible, though. This includes the lighting, which is approximated well in this example. It also serves to illustrate that opposing lighting directions can lead to similar results as the rendering in Fig. 11.4d would look very similar if the light would be rotated by 180° . These cases are hard to solve without more information and as we specifically restrict our inputs to a single image, our method fails in these situations in regard to the lighting directions.

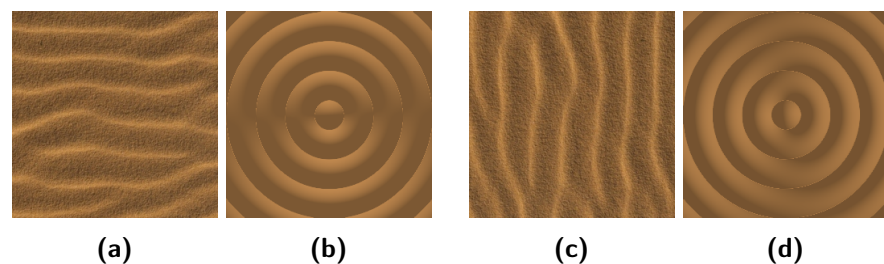


Figure 11.4 – Example of matched lighting. Rotating the image leads to a rotation in the lighting as well. The direction is matched well, although the model itself is ill-equipped to approximate the geometric structure.

11.2 LIMITATIONS

Although good results can be achieved, some limitations exist. Retrieving the lighting directions for ambiguous cases was mentioned before, but there are other limitations that should be discussed.

11.2.1 *Abilities of the Model*

Since we allow all kinds of patterns in our method, not all of them work equally well. If the model can not generate the surface structures needed to approximate the input, the results will be insufficient. We have included the *Voronoi* patterns in our results as a toy example to illustrate this point. It generates plateaus of different heights. As the way we light the height field only considers the surface normals based on the slope and not the actual height, most of the image is lit in a constant manner. Images of almost constant value are the result and are usually bad matches for the input. The *Voronoi* model is an example of a badly designed procedural regarding our specific application. It could be improved by using the underlying distances between points not only for determining the half-space the pixel position belongs to, but by mapping the distance to height in order to create varying height information.

The results show that a good match for our set of inputs can usually be found across the implemented models. Even though some models perform poorly in some situations, by comparing the match distances we can determine the model most suitable for the input and ignore models that are less appropriate.

11.2.2 *Size of the Parameter Space*

As outlined in preceding chapters, we sample the parameter space for each model and store resulting descriptors. The size of that parameter space is of importance as a bigger space requires more samples. By uniformly sampling each dimension, one quickly runs into a problem, which is known as *curse of dimensionality*. The number of samples needed to maintain the same sample density grows exponentially with the number of dimensions of the space. This leads to a substantial increase in required computing power — or computing time — if a new parameter is added to the model. A way around increased computational expenses is decreasing the sample density, but that will lead to worse matches. As noted by Gieseke et al. [Gie+14], the sample-and-retrieve approach can not find local minima like continuous methods can. Only the samples are available as potential matches and decreasing their density will also decrease the chance to find a good match.

The aforementioned observation is true for work by Gieseke et al. as well. Based on the computational capacities, a limit for the number of parameters has to be set in order to be able to sample the respective dimensions sufficiently densely. This is true even if the precomputation is considered a one-time investment. For our work the problem intensifies as two additional parameters have been introduced for the shading related parts of the process. Both the light direction, as well as the amplitude of the height field need to be added to the parameter space, effectively decreasing the number of parameters allowed for the procedural that generates the height field.

As can be seen in Fig. 11.3, a simple user intervention can help the system to create better matches. In this case an additional parameter for stretching the texture in y-direction would render this required input obsolete, but new parameters are costly. It is thus important to find a trade-off between an automatic solution as proposed, which is still computationally efficient, and required manipulations done by an artist who needs to know the advantages and the limitations of the system well. Creative solutions regarding the number or nature of the parameters chosen are needed. In this particular case, adding a parameter for the stretching in y-direction would create redundancy in the resulting images as the two parameters combined would add an additional frequency feature that can increase or contradict existing scaling. It would be better to keep the stretching as one parameter that might extend its range below zero so that those values could be used for the stretching in y-direction — effectively creating a parameter for anisotropy.

In order to improve the sampling, one has to consider the range of the sampled values as well as their distribution. Distances in parameter space are not proportional to distances between the resulting descriptors. Ideally, the samples should be distributed so that the resulting patterns show equal perceived differences. To properly create a model that is sampled this way, a study would have to be conducted that analyzes the perceived differences of patterns based on differences in the parameters used. This introduces a significant overhead as it has to be done for each model if one is serious about computing and storing as little samples as possible while still properly sampling the entire space. This is a one-time investment, though, and might be well worth the effort if the time that is saved by this approach leads to budget and time savings. Uniform sampling was applied for all our models.

11.2.3 *Quality of Shading Parameters*

The retrieved albedo depends on a specific lighting environment to produce the results that are intended. Since the radiance that is received by an observer is based on both the light arriving at a surface,

as well as the surface's scattering properties, the retrieved albedo is tied to a particular environment. As described in Chapter 7, we assume the lighting environment to be neutral and untinted. The retrieved albedo should be regarded as a neutral albedo for that reason. Deviating too much from this environment will impair the usability of the retrieved parameters.

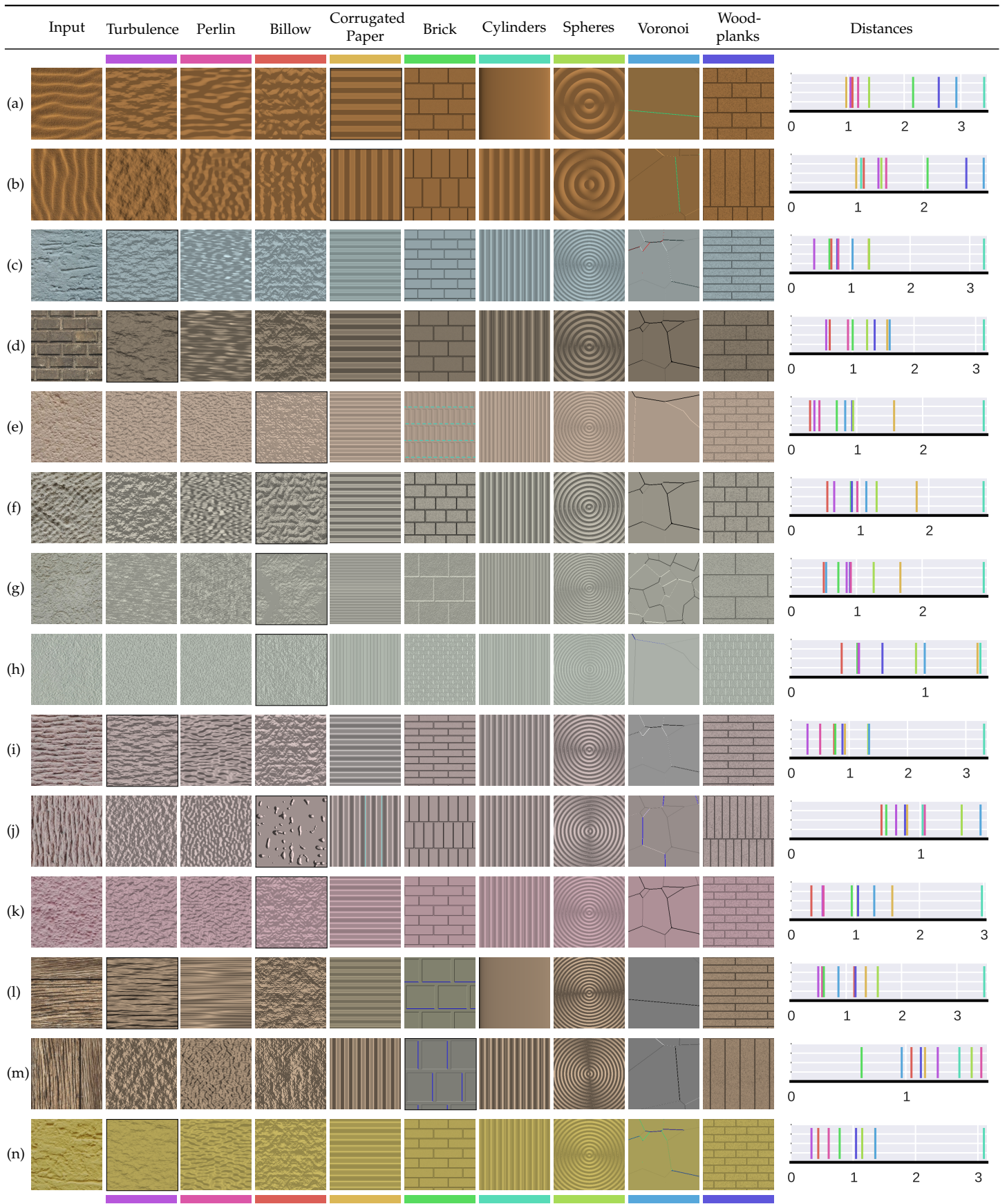


Table 4 – Complete list of matches for all implemented procedural models with colored input images. The distance plots show the distance between the descriptors of input image and matches. It is a measure of how well the model performed for the given input. The best match according to the distance metric is framed.

Part IV

CONCLUSION

FUTURE WORK

While the retrieved procedural materials are good matches for the given input images, additional techniques can be developed to improve the system. These ideas are outlined below.

12.1 MORE COMPLEX BRDFS

The Lambertian BRDF that we have used is able to model diffuse surfaces. The reflection properties of most surfaces in nature include specular components, though. A natural extension of our work will be to explore whether the geometric structure of surfaces can still be approximated well if the surface is reflective. Additionally, the parameters that model the specular reflection could be retrieved based on the given input image.

An approach that attempts both these tasks could be built on our method. If one is able to properly separate diffuse and specular reflection in the input, the diffuse component could be used as described without change. Different ways to separate the two contributions could be employed. If one is willing to invest more time in the capturing stage of the process, polarization filters can be used to separate diffuse from specular reflection [Wol89]. Additionally, entirely image-based methods exist for this task¹. The size of specular highlights in an image usually depends on both the structure of the surface (fine details break up the highlight) as well as the reflection properties controlled by a BRDF. Knowing the first should enable us to approximate the latter. Assuming a particular surface structure, adding a set of Gaussian kernels with different standard deviation σ to our filter bank could allow us to derive the shading parameters for the specular BRDF. These additional kernels are used on the specular component only.

This approach would increase the computation time because two descriptors need to be computed. It would also increase the size of the resulting descriptor that is stored. How much this overhead impacts the feasibility of precomputation and the speed of retrieval needs to be evaluated.

12.2 CAPTURING ADDITIONAL INFORMATION

Investing more time during capture could have other benefits as well. Including a Macbeth ColorChecker [MMD76] in the image would al-

¹ See Shafer [Sha85], for example

low the retrieval of information about the environment in which the image is taken. Using the chart to remove the respective tint from the image would enable us to retrieve an albedo that works for a neutral environment instead of a very specific but unknown one.

It should also be evaluated if knowing the size of the chart would allow to infer the actual amplitude of the surface structures shown in the input image.

12.3 EVALUATION

To confirm the impression that the Gabor descriptor is suitable to mimic human perception for our specific application, it should be analyzed whether a human observer would chose the same, or very similar, matches from our set of samples. This study could be conducted by allowing the test subjects to tune the parameters via sliders that allow only the values used in our sample set. Additionally, continuous sliders should be used to test whether there is a local minimum that our sampling strategy misses. These tests would give us a better idea of the performance regarding the quality of the achieved matches.

As mentioned in 9.3, evaluating whether using CIELAB for matching first order statistics could give some more insights into the differences between the results achieved using the respective spaces. Similarly, it should be tested whether we obtain different results, if we design the comparison space around sRGB instead of linear values. The latter would require to recreate all the caches and is a costly test for that reason.

12.4 SUMMARY

We have shown that it is possible to extend the work by Gieseke et al. [Gie+14] to retrieve structural information about the surface even if input images were taken in an uncontrolled lighting environment. The developed techniques can automatically retrieve parameters for a procedural displacement map without the need for human intervention. This is valuable as procedural textures often have non-intuitive parameter spaces. Additionally, the dominant direction of the lighting can be retrieved in the majority of cases and the obtained albedo allows approximating the material further. We have thus developed an interactive system that reproduces surface appearance with minimal inputs.

BIBLIOGRAPHY

- [Ans73] Francis J Anscombe. "Graphs in statistical analysis." In: *The American Statistician* 27.1 (1973), pp. 17–21.
- [AGBoo] Anthony A Apodaca, Larry Gritz, and Ronen Barzel. *Advanced RenderMan: Creating CGI for motion pictures*. Morgan Kaufmann, 2000.
- [ASoo] Michael Ashikhmin and Peter Shirley. "An anisotropic phong BRDF model." In: *Journal of graphics tools* 5.2 (2000), pp. 25–32.
- [Beh+11] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. "Cython: The best of both worlds." In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39.
- [Bevo3] Jason Bevins. *libnoise, A portable, open-source, coherent noise-generating library for C++*. 2003 – 2007. URL: <http://libnoise.sourceforge.net/> (visited on 04/10/2016).
- [BD04] Eric Bourque and Gregory Dudek. "Procedural texture matching and transformation." In: *Computer Graphics Forum*. Vol. 23. 3. Wiley Online Library. 2004, pp. 461–468.
- [BL08] Brent Burley and Dylan Lacewell. *Ptex: Per-face Texture Mapping for Production Rendering (Slides)*. 2008. URL: <http://disney-animation.s3.amazonaws.com/technology/opensource/ptex/ptex-slides.pdf> (visited on 04/10/2016).
- [Cat74] Edwin Catmull. *A subdivision algorithm for computer display of curved surfaces*. Tech. rep. DTIC Document, 1974.
- [Chao2] Moses S Charikar. "Similarity estimation techniques from rounding algorithms." In: *Proceedings of the 34th annual ACM symposium on theory of computing*. ACM. 2002, pp. 380–388.
- [CT81] Robert L Cook and Kenneth E Torrance. "A reflectance model for computer graphics." In: *ACM Siggraph Computer Graphics*. Vol. 15. 3. ACM. 1981, pp. 307–316.
- [Coo+07] Robert L Cook, John Halstead, Maxwell Planck, and David Ryu. "Stochastic simplification of aggregate detail." In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 79.

- [Coro7] Rudy Cortes. *The RenderMan Shading Language Guide*. Cengage Learning, 2007.
- [DT05] Navneet Dalal and Bill Triggs. "Histograms of Oriented Gradients for Human Detection." In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2005, pp. 886–893.
- [DeC97] Lawrence T DeCarlo. "On the meaning and use of kurtosis." In: *Psychological methods* 2.3 (1997), p. 292.
- [DG97] J-M Dischler and Djamchid Ghazanfarpour. "A procedural description of geometric textures by spectral and spatial analysis of profiles." In: *Computer Graphics Forum*. Vol. 16. 3. Wiley Online Library. 1997.
- [DBBo6] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced global illumination*. CRC Press, 2006.
- [Ebeo3] David S Ebert. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.
- [Eis+10] Christian Eisenacher, Chuck Tappan, Brent Burley, Daniel Teece, and Arthur Shek. "Example-based texture synthesis on Disney's tangled." In: *ACM SIGGRAPH 2010 Talks*. ACM. 2010, p. 32.
- [Eis+13] Christian Eisenacher, Gregory Nichols, Andrew Selle, and Brent Burley. "Sorted deferred shading for production path tracing." In: *Computer Graphics Forum*. Vol. 32. 4. Wiley Online Library. 2013, pp. 125–132.
- [Fai13] Mark D Fairchild. *Color appearance models*. John Wiley & Sons, 2013.
- [Fre61] Herbert Freeman. "On the encoding of arbitrary geometric configurations." In: *IRE Transactions on Electronic Computers* 2 (1961), pp. 260–268.
- [Fuc+07] Martin Fuchs, Volker Blanz, Hendrik Lensch, and Hans-Peter Seidel. "Adaptive sampling of reflectance fields." In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 2. ACM, 2007, p. 10.
- [Gal+12] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. "Gabor noise by example." In: *ACM Transactions on Graphics (TOG)*. Vol. 31. 4. ACM, 2012, p. 73.
- [Gie+14] L Gieseke, S Koch, J-U Hahn, and M Fuchs. "Interactive Parameter Retrieval for Two-Tone Procedural Textures." In: *Computer Graphics Forum*. Vol. 33. 4. Wiley Online Library. 2014, pp. 71–79.

- [GD10] Guillaume Gilet and J-M Dischler. “An Image-Based Approach for Stochastic Volumetric and Procedural Details.” In: *Computer Graphics Forum*. Vol. 29. 4. Wiley Online Library. 2010, pp. 1411–1419.
- [GDG12] Guillaume Gilet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. “Multiple kernels noise for improved procedural texturing.” In: *The Visual Computer* 28.6-8 (2012), pp. 679–689.
- [GS+84] Clark R Givens, Rae Michael Shortt, et al. “A class of Wasserstein metrics for probability distributions.” In: *The Michigan Mathematical Journal* 31.2 (1984), pp. 231–240.
- [GW07] Rafael C Gonzalez and Richard E Woods. *Digital image processing 3rd edition*. Prentice Hall, 2007.
- [HVS13] Christophe Hery, Ryusuke Villemin, and Pixar Animation Studios. *Physically Based Lighting at Pixar*. 2013.
- [Hof+10] Naty Hoffman, Yoshiharu Gotanda, Adam Martinez, and Ben Snow. “Physically based shading models for film and game production.” In: *SIGGRAPH Course* (2010).
- [HBC10] Rui Hu, Mark Barnard, and John Collomosse. “Gradient field descriptor for sketch based retrieval and localization.” In: *2010 17th IEEE International Conference on Image Processing*. IEEE. 2010, pp. 1025–1028.
- [ISO04] ISO, Geneva, Switzerland. *Photography and graphic technology – Extended colour encodings for digital image storage, manipulation and interchange – Part 1: Architecture and requirements*. Norm. 2004.
- [ISO08] ISO, Geneva, Switzerland. *Colorimetry — Part 4: CIE 1976 L*a*b* Colour space*. Norm. 2008.
- [Inc15] Image Engine Design Inc. *Chappie*. 2015. URL: <http://image-engine.com/film/chappie/> (visited on 04/10/2016).
- [Jer77] Abdul J Jerri. “The Shannon sampling theorem—Its various extensions and applications: A tutorial review.” In: *Proceedings of the IEEE* 65.11 (June 1977), pp. 1565–1596.
- [JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: <http://www.scipy.org/> (visited on 04/10/2016).
- [Kaj86] James T Kajiya. “The rendering equation.” In: *ACM Siggraph Computer Graphics*. Vol. 20. 4. ACM. 1986, pp. 143–150.

- [Ker07] Robert Kern. *A Simple File Format for NumPy Arrays*. 2007. URL: <http://docs.scipy.org/doc/numpy/neps/npv-format.html> (visited on 04/10/2016).
- [KR04] Reinhard Klette and Azriel Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Elsevier, 2004.
- [Koe84] Jan J Koenderink. "The structure of images." In: *Biological cybernetics* 50.5 (1984), pp. 363–370.
- [Lag+10a] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S Ebert, JP Lewis, Ken Perlin, and Matthias Zwicker. "A survey of procedural noise functions." In: *Computer Graphics Forum*. Vol. 29. 8. Wiley Online Library. 2010, pp. 2579–2600.
- [Lag+10b] Ares Lagae, Peter Vangorp, Toon Lenaerts, and Philip Dutré. "Procedural isotropic stochastic textures by example." In: *Computers & Graphics* 34.4 (2010), pp. 312–321.
- [Lag+10c] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony Derose, George Drettakis, David S Ebert, JP Lewis, Ken Perlin, and Matthias Zwicker. "State of the art in procedural noise functions." In: *EG 2010-State of the Art Reports*. The Eurographics Association. 2010.
- [Lew94] Robert R Lewis. "Making shaders more physically plausible." In: *Computer Graphics Forum*. Vol. 13. 2. Wiley Online Library. 1994, pp. 109–120.
- [Lin+06] Wen-Chieh Lin, James Hays, Chenyu Wu, Yanxi Liu, and Vivek Kwatra. "Quantitative evaluation of near regular texture synthesis algorithms." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 2006, pp. 427–434.
- [Lin+11] Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour, Kai Yu, Liangliang Cao, and Thomas Huang. "Large-scale image classification: fast feature extraction and svm training." In: *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 2011, pp. 1689–1696.
- [Lin13] Tony Lindeberg. *Scale-space theory in computer vision*. Springer Science & Business Media, 2013.
- [MM96] Bangalore S Manjunath and Wei-Ying Ma. "Texture features for browsing and retrieval of image data." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 18. 8. IEEE, 1996, pp. 837–842.

- [Mar80] S Marçelja. "Mathematical description of the responses of simple cortical cells." In: *JOSA* 70.11 (1980), pp. 1297–1300.
- [McA+12] Stephen McAuley, Stephen Hill, Naty Hoffman, Yoshiharu Gotanda, Brian Smits, Brent Burley, and Adam Martinez. "Practical physically-based shading in film and game production." In: *ACM SIGGRAPH 2012 Courses*. ACM. 2012, p. 10.
- [MMD76] Calvin S McCamy, H Marcus, and JG Davidson. "A color-rendition chart." In: *J. App. Photog. Eng* 2.3 (1976), pp. 95–99.
- [Olio6] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.
- [ON94] Michael Oren and Shree K Nayar. "Generalization of Lambert's reflectance model." In: *Proceedings of the 21st Annual Conference on Computer graphics and Interactive Techniques*. ACM. 1994, pp. 239–246.
- [Per85] Ken Perlin. "An image synthesizer." In: *ACM Siggraph Computer Graphics* 19.3 (1985), pp. 287–296.
- [PH04] Matt Pharr and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004.
- [Pho75] Bui Tuong Phong. "Illumination for computer generated pictures." In: *Communications of the ACM* 18.6 (1975), pp. 311–317.
- [Pou11] Foteini Tania Pouli. "Statistics of image categories for computer graphics applications." PhD thesis. University of Bristol, 2011.
- [PD05] V Shiv Naga Prasad and Justin Domke. "Gabor filter visualization." In: *J. Atmos. Sci* 13 (2005).
- [Rei+01] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. "Color transfer between images." In: *IEEE Computer graphics and applications* 5 (2001), pp. 34–41.
- [RFS] Zachary Repasky, Patrick Schork Kevin McNamara Susan Fong, and Pixar Animation Studios. *Large Scale Geometric Visibility Culling on Brave*. URL: <http://graphics.pixar.com/library/VisibilityCulling/paper.pdf> (visited on 04/10/2016).

- [Rob05] Michael Robin. *Gamma correction*. 2005. URL: https://web.archive.org/web/20090531130845/http://broadcastengineering.com:80/newsrooms/broadcasting_gamma_correction/ (visited on 04/10/2016).
- [RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval." In: *International journal of computer vision* 40.2 (2000), pp. 99–121.
- [Sey15] Mike Seymour. *Mazing Scorched effects*. 2015. URL: <https://www.fxguide.com/featured/mazing-scorched-effects/> (visited on 04/10/2016).
- [Sha85] Steven A Shafer. "Using color to separate reflection components." In: *Color Research & Application* 10.4 (1985), pp. 210–218.
- [Smi95] Alvy Ray Smith. "A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square!" In: *Microsoft Computer Graphics, Technical Memo 6* (1995).
- [Smi+97] Steven W Smith et al. *The scientist and engineer's guide to digital signal processing*. California Technical Pub. San Diego, 1997.
- [Ste61] Stanley Smith Stevens. "To honor Fechner and repeal his law." In: *Science* 133 (1961), pp. 80–133.
- [VDWCV11] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation." In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.
- [WSM11] Chun-Po Wang, Noah Snavely, and Steve Marschner. "Estimating dual-scale properties of glossy surfaces from step-edge lighting." In: *ACM Transactions on Graphics (TOG)*. Vol. 30. 6. ACM. 2011, p. 172.
- [War12] Colin Ware. *Information visualization: perception for design*. Elsevier, 2012.
- [Wei+09] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. "State of the art in example-based texture synthesis." In: *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association. 2009, pp. 93–117.
- [Wil83] Lance Williams. "Pyramidal parametrics." In: *ACM Siggraph Computer Graphics*. Vol. 17. 3. ACM. 1983, pp. 1–11.

- [Wol89] Lawrence B Wolff. "Using polarization to separate reflection components." In: *1989 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 1989, pp. 363–369.

DECLARATION

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, April 12th 2016

Matti Grüner

DEKLARATION

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, April 12th 2016

Matti Grüner